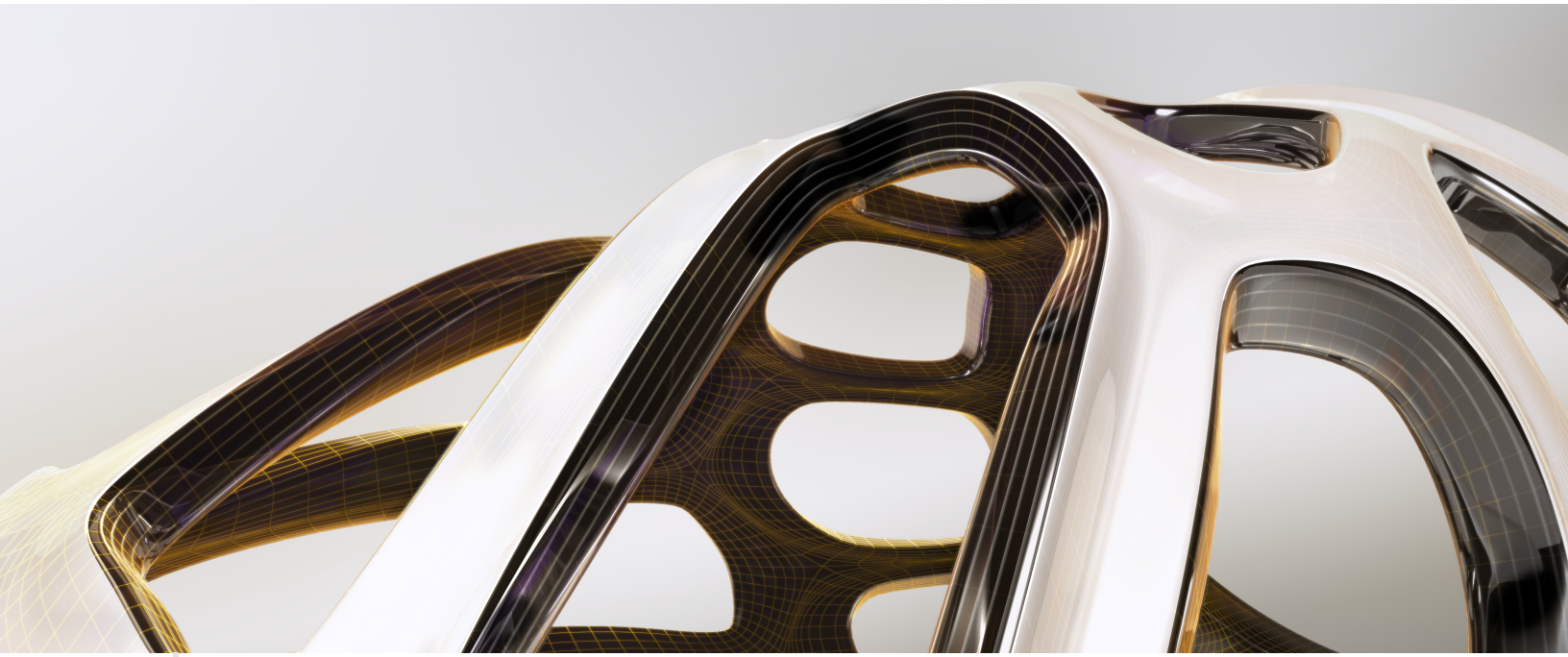
AUTODESK.



Post Processor Manual

Works with:
HSMWorks
Inventor HSM Express
Inventor HSM
Inventor HSM Pro

Autodesk Post Processor Manual

1. Scope
2. Intended audience
3. Skill level of audience
4. History
5. Why we need posts
6. Overview of post processor workings
7. Basic post processor functions
8. Overview of a simple X and Y axis post
9. Testing a post
10. Editing a post

Scope

This manual is intended for those who wish to makes their own edits to existing library posts. It is also for those who wish to expand the functionality of their working posts.

Intended Audience

The instructions you find here are intended for machinists with some programming experience, and CAM engineers looking to provide edit-free CNC programs for the machine shop.

Skill level

It is assumed that the reader has a working knowledge of Inventor HSM Express. If you have not done so already, review the programming examples that came with your CAM package. You will need this to produce working CAM programs.

History

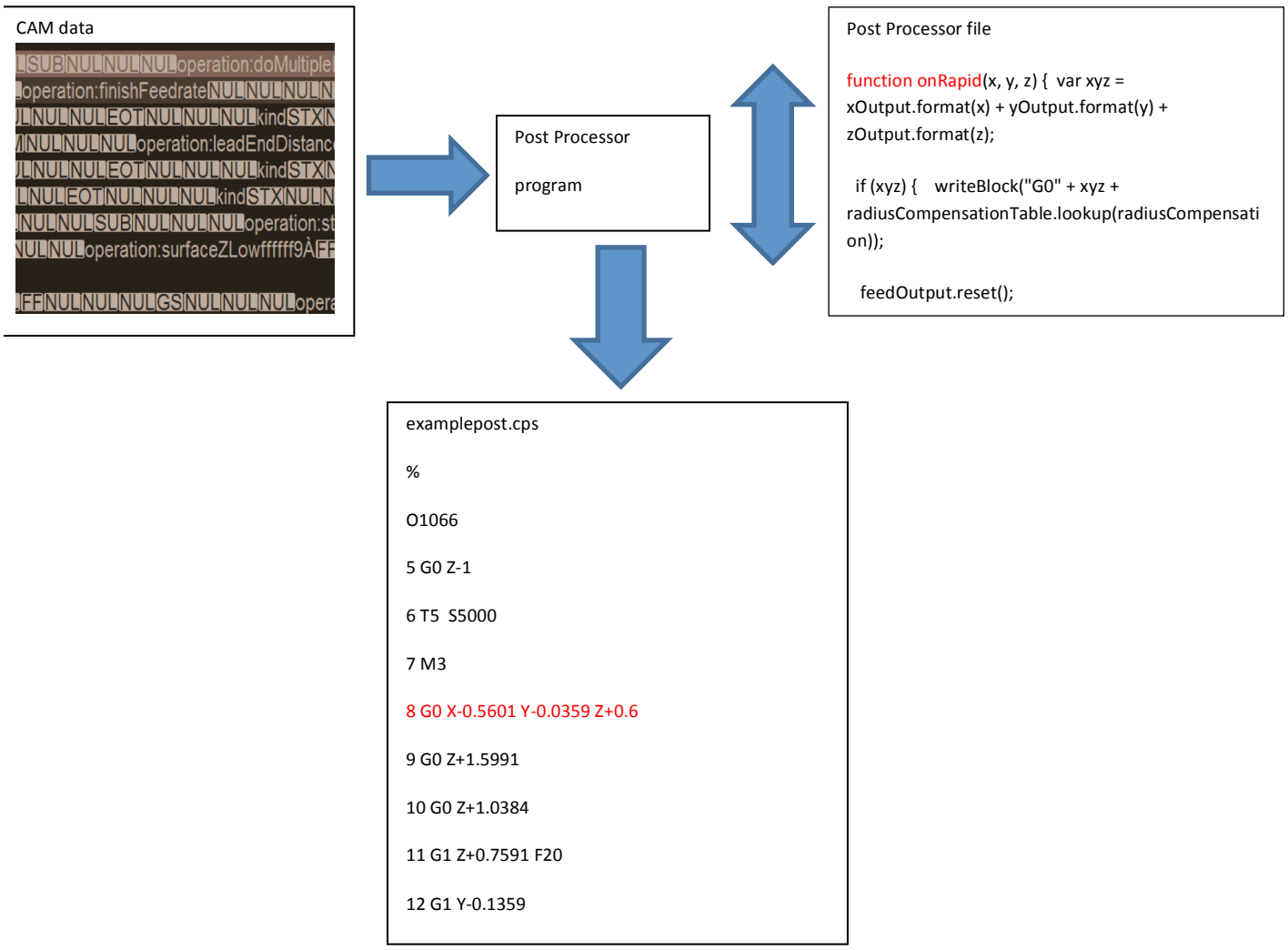
The first CAM software packages came with an executable program that was run on a computer to convert the CAM instructions into G and M code. G&M code was developed as a language to make machines move. Initially the code consisted of line and arc moves, but as time went on it became more sophisticated and evolved into many variations of a standard ISO code.

Why we need posts

The need for an interface program became necessary as G&M became more complicated and evolved into more complex variations on each machine tool. More machine capabilities like work fixture offsets, radius compensation, tool length compensation and automatic tool changers were added. All these features vary from machine to machine so some kind of editable interface had to be made.

Overview

The CAM program makes a non-readable intermediate data file from the instructions in the CAM program. These are very generic and have no particular format. They contain positions in X, Y, and Z, and functions like the tool data and the type of machining operation (threading, drilling, etc.) The post processor program reads this data and looks for the relevant section in the post processor file. When a relevant section is found, all the instructions and logic are followed in that section. The post processor then continues to read the intermediate file until it reaches the end.



Basic Functions

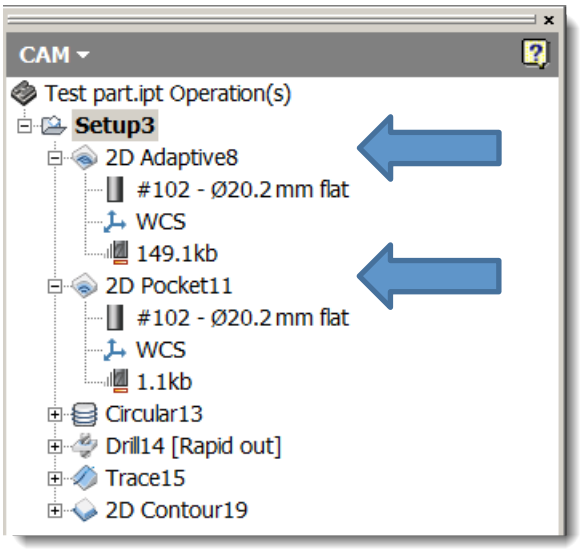
The post processor file has an extension .cps which is based on Java Script. The very core of the post processor is comprised of sections that are interpreted in a linear sequence from the beginning of the intermediate file until it reaches the end. Once the intermediate file has ended, it is deleted. These sections are listed below:

onOpen

Starts the beginning of the NC program with items like the program Name for storage in the NC control memory.

onSection

This section is interpreted for every OPERATION in the SETUP contained in the CAM program.



This could be the work fixture offset used, the tool, any subsequent tool for pre-selection, and coolant codes. Also movements to a safe approach position, before the actual machining moves.

onMovement

Handles feed by the movement type. This can also be done in the onRapid(), onLinear(), and onCircular() sections.

onRadiusCompensation

Commands used for radius compensation mode. This can also be done in the onRapid(), onLinear(), and onCircular() sections.

onRapid

Commands for rapid traverse motion. Example G0.

onLinear

Commands for linear feed motion. Example G01.

onCircular

Commands for output circular motion, if supported. Example G02 G03.

onRapid5D

Commands for output of rapid 5-axis traverse motion.

onLinear5D

Commands for output of linear 5-axis feed motion.

onDwell

Commands for output of a dwell time in the NC code.

onSpindleSpeed

Commands for output of spindle speed.

onCycle

Commands for NC “canned” cycles, like drilling and threading, that are executed on various points in the CAM program.

onCyclePoint

Commands for output of the various points from the CAM system for the canned cycle to use.

onCycleEnd

End and reset the canned cycle.

onSectionEnd

Commands to retract the machine to a safe position and turn off modal codes. This function is called at the end of each section, and may be useful to add code for tool break control.

onClose

Commands to append to the end of the NC program.

onTerminate

Allows access to the output file for external applications. The output file will be locked and not fully closed until onTerminate() is invoked. The log file is still locked at this point.

Basic post internals

Because the post processor follows the rules of Java Script we have the ability to customize the various sections of the NC program. We can have variables and control the flow of the program using logic.

To use a variable, we just tell the post the name of the variable:

```
var coolantCode = 8;
```

var stands for variable. Notice the semicolon on the end of the line.

We can also use variables to store text.

```
var coolant=”M08”;

var radiusLeft="G41";
```

Note: *radiusLeft is not the same variable as RadiusLEFT because variables are case-sensitive.*

We can also place variables on a single line.

```
var radiusLeft="G41", stop=30, coolant=”M08”;
```

To compare the values stored in variables, we use arithmetic operators.

```
+

=

var x=5;
var y=6;
var z=x+y;
```

Arithmetic operators:

Operator	Description	Example	Value of y	Resultingx value
+	Addition	x=y+2	5	7
-	Subtraction	x=y-2	5	3
*	Multiplication	x=y*2	4	8
/	Division	x=y/2	5	2.5
++	Increment	x=++y	6	6
		x=y++	6	5
--	Decrement	x=--y	4	4
		x=y--	4	5

Conditional Statements

When you write a post processor, you will want to perform actions for different reasons. You can use conditional statements in the post processor to do this.

You can make comments in the code with //. The post processor ignores everything after this.

The braces {} surround the next commands.

- **if statement** - This is used to process parts of the code only if a specified condition is true.

```
if (useCoolant){
  coolantCode="M08"; // coolant  on
}
```

- **if...else statement** – Use this to execute some code if the condition is true and some other code if the condition is false.

```
if (useCoolant){
  coolantCode="M08"; // coolant  on   continue here if useCoolant is set to true
} else {
  coolantCode="M09"; // coolant  off     continue here if useCoolant is set to false
}
```

- **if...else if....else statement** - Use this to select one of many chunks of code to be processed.

```
if (useCoolant) {
  coolantCode="M08"; // coolant  on   continue here if useCoolant is set to true
} else {
if (!useCoolant) {
  coolantCode="M09"; // coolant  off     continue here if useCoolant is set to false
} else {
 coolantCode="M07"; // coolant  mist     continue here if useCoolant is neither true or false
}
}
```

- **switch statement** - Use this statement to select one of many chunks of code to be executed.

```
switch(coolantCode) {
case 8:
  coolantCode="M08";
  break;
```

```
case 9:
 coolantCode=”M09”;
 break;
case 7;
 coolantCode=”M07”;
 default:
 coolantCode=”M61”;
}
```

Assignment operators

If we start with x=10 and y=5, here are the results:

Operator	Example	Similar to	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

Text can also be modified using operators:

```
var coolantCode=”M08”,spindleStop=“M03”
var turnoff=coolantCode  + SP + spindleStop (SP means space)
```

The variable called turnoff would become M08 M03.

If you add text and a number, the result is just text:

```
var mword=”M0”, mcode=8
coolant=mword + mcode
```

Coolant would be M08 but would contain only text.

Comparison Operators

If we start with x equaling 5:

Operator	Description	Comparing	Returns
==	is equal to	x==8	*false*
		x==5	*true*
===	is exactly equal to (value and type)	x==="5"	*false*
		x===5	*true*
!=	is not equal	x!=8	*true*
!==	is not equal (neither value nor type)	x!=="5"	*true*
		x!==5	*false*
>	is greater than	x>8	*false*
<	is less than	x<8	*true*
>=	is greater than or equal to	x>=8	*false*
<=	is less than or equal to	x<=8	*true*

Example:

```
if (workOffset > 6) var p = workOffset – 6
```

Logical Operators

Logical operators are used to find the logic between variables or values.

If **x=4** and **y=7**, the table below explains the logical operators:

Operator	Description	Example
&&	And	(x < 10 && y > 1) is true
||	Or	(x==4 || y==5) is false
!	Not	!(x==y) is true

Example:

```
if ((programId >= 8000) && (programId <= 9999))
```

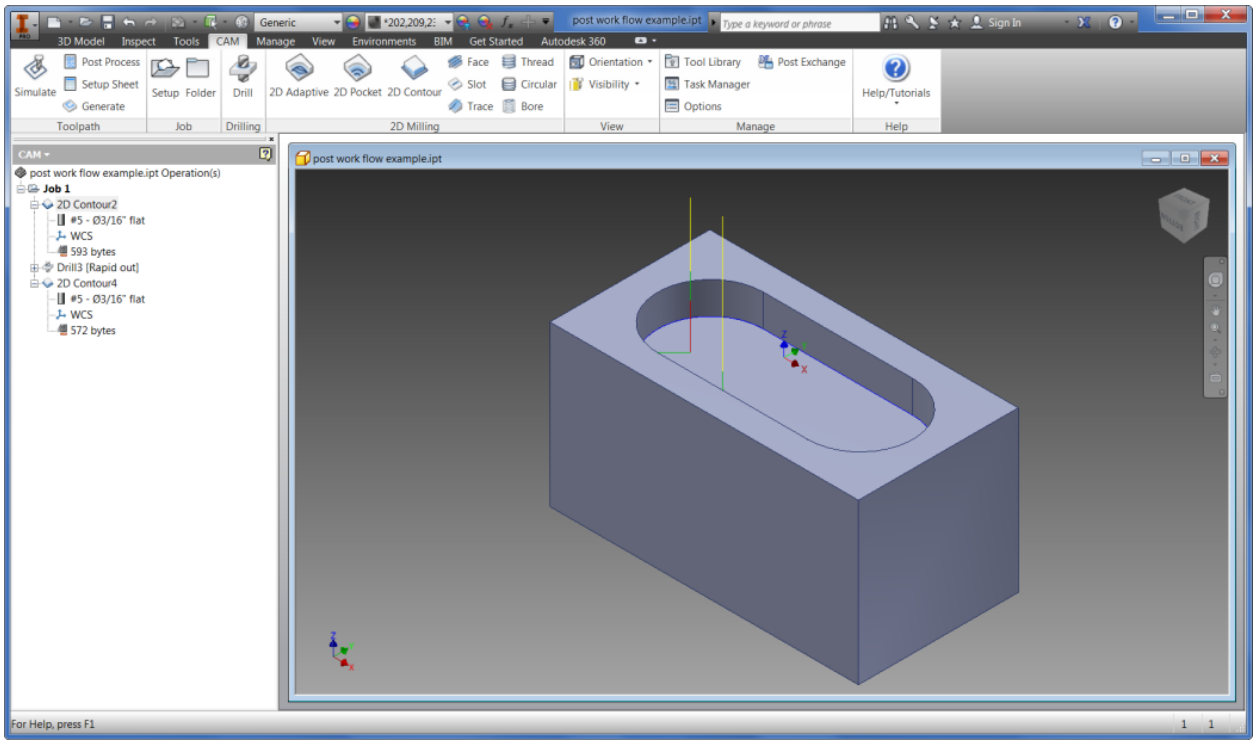
This statement is true if program id is more than 8000 and less than 9999

My First Post

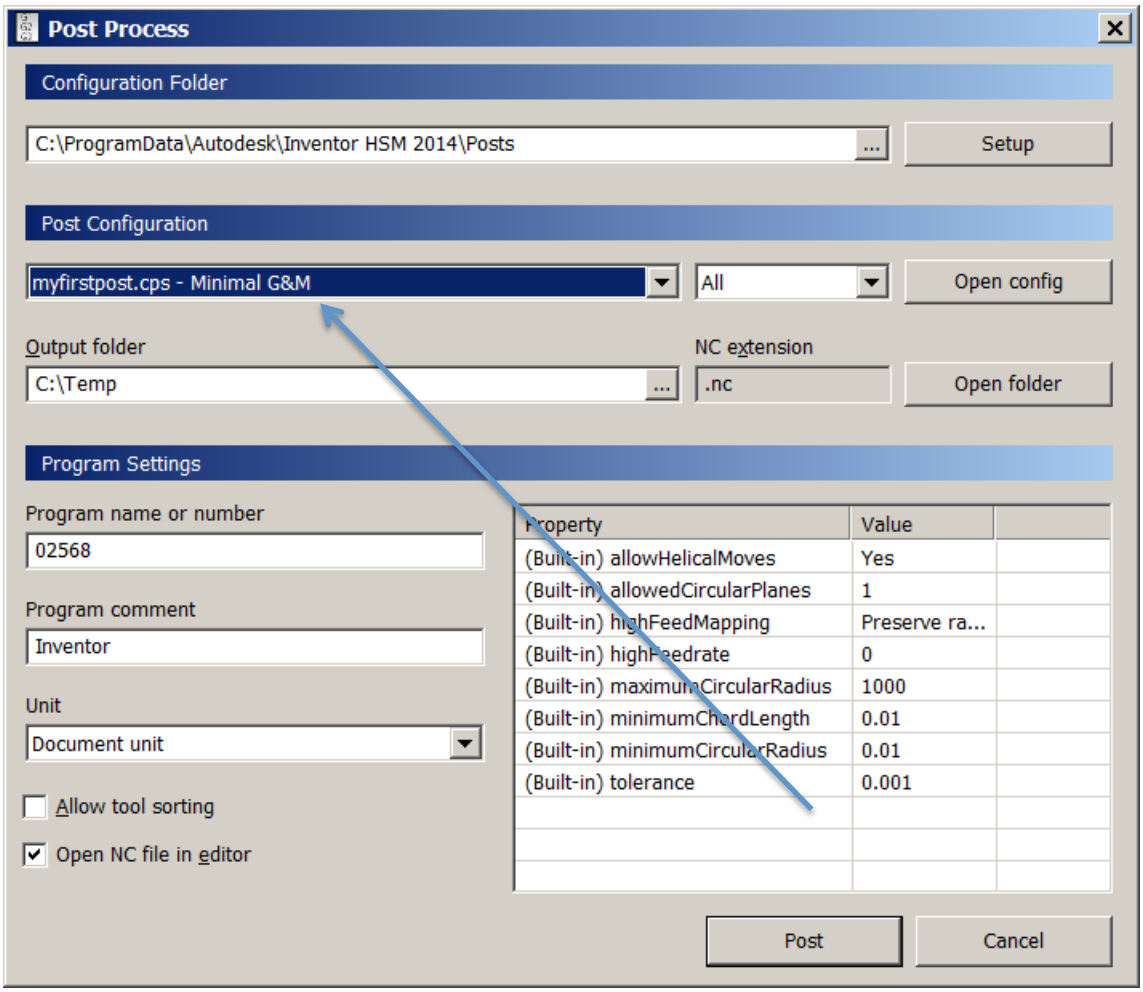
Now let's make a post. First we will create a new post file using a text editor like Notepad. Any file with the extension CPS is automatically recognized by the HSM Expresss Editor called myfirstpost.cps in a folder on your hard drive. We will include the function onOpen which the post software knows to process at the beginning of the NC program.

```
function onOpen() {
  writeln("%");
  writeln((programName ? (programName) : "") );
}
```

Open the sample CAM part (post work flow example.ipt) and select the first operation in the CAM Browser, and then select Post Process to produce the NC output.



The only thing we did in onOpen is to send a % to the function writeln. Then on a new line we sent the program number from the post dialog box to the function writeln.



When you click the Post button you should get this NC file in the output folder. Now if you open the NC file you will see:

```
%
02568
```

Very cool, you just made you first piece of NC code! Okay, it's just the percent sign and the program number but it's a start.

Most controls have a limit on the length of the NC file name. Later we will control this.

Now let's add the CAM program tool information for the first operation with onSection.

```
var blockNumber = 5;
//writes the specified block of G code.
function writeBlock(block) {
  writeln( "N" + blockNumber + SP + block);
  ++blockNumber;
}
function onOpen() {
  writeln("%");
```



```
  writeln((programName ? (programName) : "") );
}
function onSection() {
  writeBlock("T" + tool.number + SP + " S" + tool.spindleRPM);
}
```

Here is our new NC program.

```
%
02568
N5 T4 S5000
```

Note: *The Tool number might be different depending on which Operation you have chosen to post-process.*

Wow, we will be cutting metal soon!

Before onOpen we added a user defined *function* called writeBlock. We made it a *function* because we want to use it more than once.

It adds a number at the beginning of each line and then increments the number by one for the next line. We also made a variable to be used for each NC block; then we will increase it by one every time we use it.

```
// This function Writes an NC block of G code.
function writeBlock() {
  writeWords2("N" + blockNumber, arguments);
  ++blockNumber;
}
```

Notice the use of // to make a comment that the post program ignores.

Now we can add some movement to our NC program. This will move the tool to the initial position before cutting in the operation.

```
var blockNumber = 5;

// This function Writes an NC block of G code.
function writeBlock() {
  writeWords2("N" + blockNumber, arguments);
  ++blockNumber;
}

function onOpen() {
  writeln("%");
  writeln((programName ? (programName) : "") );
}
```

```
function onSection() {
  var initialPosition = getFramePosition(currentSection.getInitialPosition());
  writeBlock("T" + tool.number + SP + " S" + tool.spindleRPM);
  writeBlock("G0 " + initialPosition.x + initialPosition.y + initialPosition.z);
}
```

Now if we click post, we will get NC code that looks like this.

```
%
02568
N5 T4 S5000
N6 G0-0.56012596671036870.0078492124718943911.6400000054066575
```

The X, Y, and Z positions are raw data from the CAM software. We will have to add some formatting commands so that the NC control can understand it.

```
var xyzFormat = createFormat({decimals:(unit == MM ? 3 : 4), forceSign:true});
var feedFormat = createFormat({decimals:(unit == MM ? 0 : 2)});

//linear Output
var xOutput = createVariable({prefix:"X"}, xyzFormat);
var yOutput = createVariable({prefix:"Y"}, xyzFormat);
var zOutput = createVariable({prefix:"Z"}, xyzFormat);
var feedOutput = createVariable({prefix:"F"}, feedFormat);
```

Okay, I know it looks complicated, but we will use it many times so it will save time in the long run. Believe me.

```
var blockNumber = 5;
// This function Writes an NC block of G code.
function writeBlock() {
  writeWords2("N" + blockNumber, arguments);
  ++blockNumber;
}
function onOpen() {
  writeln("%");
  writeln((programName ? (programName) : "") );
}
function onSection() {
  var initialPosition = getFramePosition(currentSection.getInitialPosition());
  writeBlock("T" + tool.number + SP + " S" + tool.spindleRPM);
  writeBlock("G0 " + xOutput.format(initialPosition.x) + yOutput.format(initialPosition.y) +
zOutput.format(initialPosition.z));
}
```

Look what we get! Making chips very soon I am thinking….

```
%
```

```
02568
N5 T4 S5000
N6 G0 X-0.1 Y+0.025 Z+0.64
```

Now to add some positioning code at the end of the post file for rapid movements.

```
function onRapid(x, y, z) {
  writeBlock("G0", xOutput.format(x), yOutput.format(y), zOutput.format(z));
  feedOutput.reset();
}
```

If the CAM program contains two or more operations, notice that the post is producing the rapids for all the rapids in all operations and speeds.

```
%
02568
N5 T4 S5000
N6 G0 X-0.1 Y+0.025 Z+0.64
N7 G0 Z+0.0794
N8 G0 X+0.1 Z+0.64
```

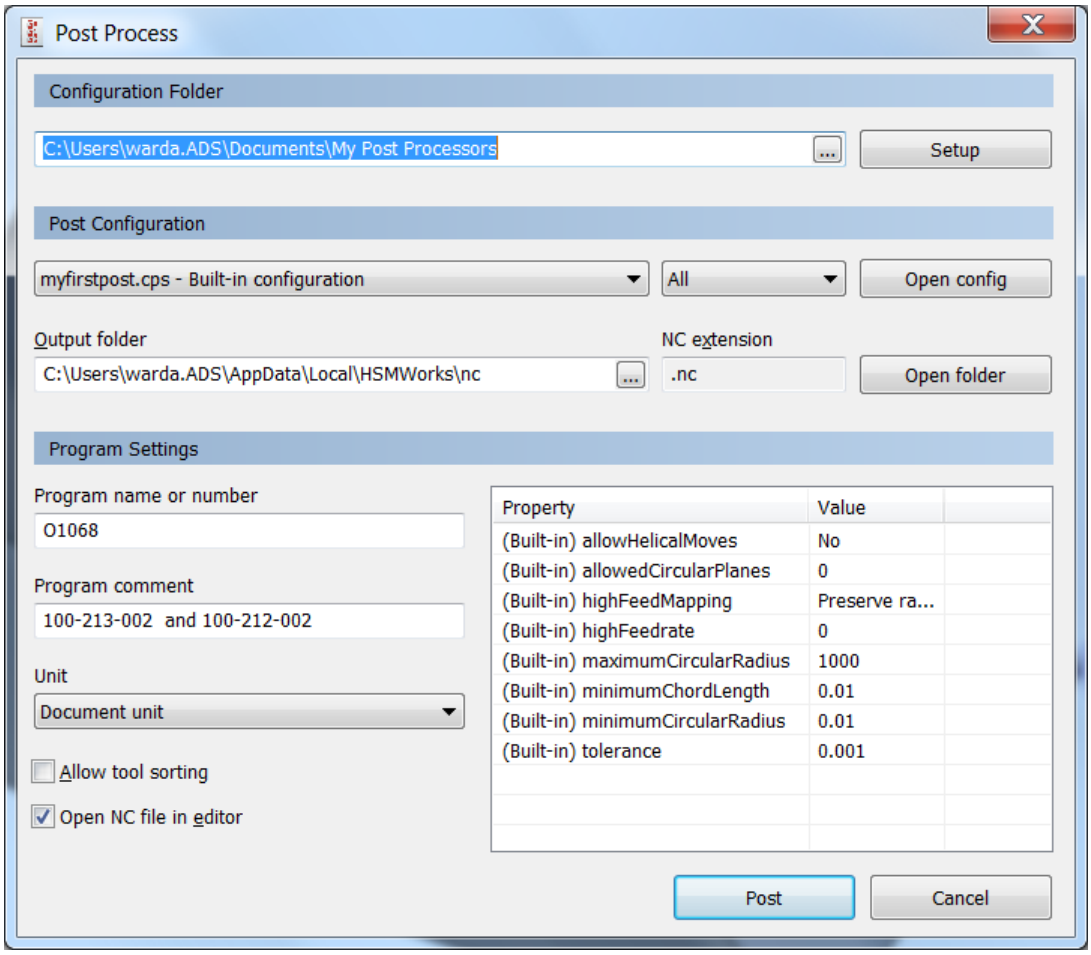
We now need to process the feed movements. Add these lines at the end of your post:

```
function onLinear(x, y, z, feed) {
  writeBlock("G1", xOutput.format(x), yOutput.format(y), zOutput.format(z),
feedOutput.format(feed));
}
```

Now post the code again.

```
%
02568
N5 T4 S5000
N6 G0 X-0.1 Y+0.025 Z+0.64
N7 G0 Z+0.0794
N8 G1 Z-0.2 F20
N9 G1 X-0.2375 Y-0.075
N10 G1 X-0.2373 Y-0.0835
N11 G1 X-0.2369 Y-0.0919
N12 G1 X-0.2361 Y-0.1004
N13 G1 X-0.2351 Y-0.1088
N14 G1 X-0.2337 Y-0.1172
N15 G1 X-0.2321 Y-0.1255
N16 G1 X-0.2301 Y-0.1337
N17 G1 X-0.2279 Y-0.1419
N18 G1 X-0.2253 Y-0.15
N19 G1 X-0.2225 Y-0.158
```

I know you are thinking there are way too many feed moves for the program. That's because the post program will detect arc movements and if there is no section to deal with this, it will just break down the arcs into lots of small feed moves. You can change the length of the Chord it uses by changing the (Built-in) tolerance property.



Reduce its value from 0.001 to 0.01, and note the reduction of the number of lines.

Now we will add the code at the end of the post to read arc movements from the CAM program.

```
function onCircular(clockwise, cx, cy, cz, x, y, z, feed) {
  var start = getCurrentPosition();
  var f = feedOutput.format(feed);
  writeBlock("G" + (clockwise ? 2 : 3), xOutput.format(x), yOutput.format(y), zOutput.format(z),
iOutput.format(cx - start.x, 0), jOutput.format(cy - start.y, 0) , f);   }
```

After posting it, you will notice absolutely no difference. That's because the post needs some settings for its own use to calculate arc movements.

Now add these lines at the beginning of your post file:

```
minimumChordLength = spatial(0.01, MM);
minimumCircularRadius = spatial(0.01, MM);
maximumCircularRadius = spatial(1000, MM);
minimumCircularSweep = toRad(0.01);
maximumCircularSweep = toRad(180);
allowHelicalMoves = true;
allowedCircularPlanes = udefined; // allow any circular motion
```

And add these formatting lines after the formatting code for linear moves:

```
//circular output
var iOutput = createReferenceVariable({prefix:"I"}, xyzFormat);
var jOutput = createReferenceVariable({prefix:"J"}, xyzFormat);
var kOutput = createReferenceVariable({prefix:"K"}, xyzFormat);
```

Aha! You are saying that looks more like an NC program!

```
%
02568
N5 T4  S5000
N6 G0 X-0.1 Y+0.025 Z+0.64
N7 G0 Z+0.0794
N8 G1 Z-0.2 F20
N9 G1 X-0.2375 Y-0.075
N10 G3 X+0 Y-0.3125 I+0.2375
N11 G1 X+0.5 F40
N12 G3 Y+0.3125 J+0.3125
N13 G1 X-0.5
N14 G3 Y-0.3125 J-0.3125
N15 G1 X+0
N16 G3 X+0.2375 Y-0.075 J+0.2375
N17 G1 X+0.1 Y+0.025
N18 G0 Z+0.64
```

Okay, now let's add the functionality to apply radius compensation on linear moves by changing the writeBlock.

From:

```
function onLinear(x, y, z, feed) {
  writeBlock("G1", xOutput.format(x), yOutput.format(y), zOutput.format(z),
feedOutput.format(feed));
}
```

To:

```
function onLinear(x, y, z, feed) {
  var xyz = xOutput.format(x) + " " + yOutput.format(y) + " " + zOutput.format(z);
```

```
  var f = feedOutput.format(feed);
  if (xyz) {
    writeBlock("G1_" + xyz + radiusCompensationTable.lookup(radiusCompensation) + " " + f);
  }
}
```

After this section ending…

```
maximumCircularSweep = toRad(180);
allowHelicalMoves = true;
allowedCircularPlanes = undefined; // allow any circular motion
```

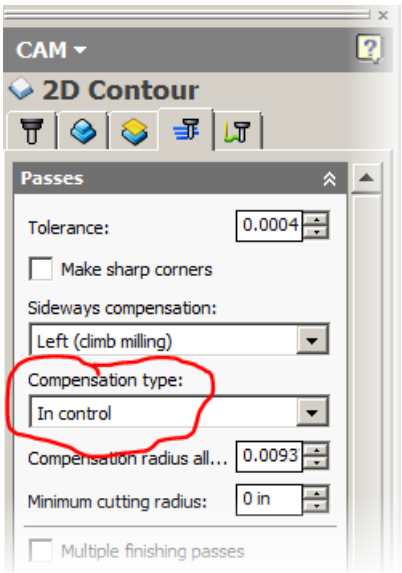
…add this code:

```
var radiusCompensationTable = new Table(
  [" G40", " G41", " G42"],
  {initial:RADIUS_COMPENSATION_OFF},
  "Invalid radius compensation"
);
```

This is a table for choosing radius compensation from the CAM program. Now post again and you should see this:

```
%
02568
N5 T4  S5000
N6 G0 X-0.1 Y+0.025 Z+0.64
N7 G0 Z+0.0794
N8 G1 Z-0.2 F20
N9 G1 X-0.2375 Y-0.075 G41
N10 G3 X+0 Y-0.3125 I+0.2375
N11 G1 X+0.5 F40
N12 G3 Y+0.3125 J+0.3125
N13 G1 X-0.5
N14 G3 Y-0.3125 J-0.3125
N15 G1 X+0
N16 G3 X+0.2375 Y-0.075 J+0.2375
N17 G1 X+0.1 Y+0.025 G40
N18 G0 Z+0.64
%
```

Note: *The G41 and G40 will only show up if the radius compensation is set to be created by the Post Processor as shown below. This is also only an option in the 2D Contour strategy.*



Did you see the G41 and G40 codes added to the linear moves to apply radius compensation? Very cool.

We need some kind of safety move at the end of every section or CAM operation. Add this code at the end of your post file.

```
function onSectionEnd(){
  writeBlock("G0 G91 G28"); // retract
  writeBlock("G90");
}
```

```
%
02568
N5 T4 S5000
N6 G0 X-0.1 Y+0.025 Z+0.64
N7 G0 Z+0.0794
N8 G1 Z-0.2 F20
N9 G1 X-0.2375 Y-0.075 G41
N10 G3 X+0 Y-0.3125 I+0.2375
N11 G1 X+0.5 F40
N12 G3 Y+0.3125 J+0.3125
N13 G1 X-0.5
N14 G3 Y-0.3125 J-0.3125
N15 G1 X+0
N16 G3 X+0.2375 Y-0.075 J+0.2375
N17 G1 X+0.1 Y+0.025 G40
N18 G0 Z+0.64
N19 G0 G91 G28
N20 G90
```

Now we have a safety move between operations in the CAM program.

At the end of the program we need to cancel a few things out and end the NC program. Add these lines at the end of your post file:

```
function onClose() {
writeBlock("M30"); // stop program, spindle stop, coolant off
 writeln("%");
}
```

This will only be used at the end of the program. This is how posting only the first operation should look.

```
%
N5 T4  S5000
N6 G0 X-0.1 Y+0.025 Z+0.64
N7 G0 Z+0.0794
N8 G1 Z-0.2 F20
N9 G1 X-0.2375 Y-0.075 G41
N10 G3 X+0 Y-0.3125 I+0.2375
N11 G1 X+0.5 F40
N12 G3 Y+0.3125 J+0.3125
N13 G1 X-0.5
N14 G3 Y-0.3125 J-0.3125
N15 G1 X+0
N16 G3 X+0.2375 Y-0.075 J+0.2375
N17 G1 X+0.1 Y+0.025 G40
N18 G0 Z+0.64
N19 G0 G91 G28
N20 G90
N21 M30
%
```

The post software will output drilling moves without the use of functions; just like when we machined arcs without the onCircular function. To see this, just highlight the Drill1 operation in the CAM example and post again. You will get this:

```
%
02568
N5 T1  S500
N6 G0 X-0.85 Y-0.3154 Z+0.64
N7 G0 Z+0.24
N8 G0 Z+0.2
N9 G1 Z-0.2 F5.82
N10 G0 Z+0.24
N11 G0 Y+0.35
N12 G0 Z+0.2
N13 G1 Z-0.2 F5.82
```

```
N14 G0 Z+0.24
N15 G0 X+0.85
N16 G0 Z+0.2
N17 G1 Z-0.2 F5.82
N18 G0 Z+0.24
N19 G0 Y-0.3154
N20 G0 Z+0.2
N21 G1 Z-0.2 F5.82
N22 G0 Z+0.24
N23 G0 Z+0.64
N24 G0 G91 G28
N25 G90
N26 M30
%
```

This next section will add Tool Length Offset Compensation which corrects the Z height of the toolpath for tools of different lengths.

Add the following line to the bottom of the Onsection function:

```
writeBlock("G43" + SP + "H0" + tool.number);
```

The Onsection function should now look like the following:

```
function onSection() {
  var initialPosition = getFramePosition(currentSection.getInitialPosition());
  writeBlock("T" + tool.number + SP + " S" + tool.spindleRPM);
  writeBlock("G0 " + xOutput.format(initialPosition.x) + yOutput.format(initialPosition.y) +
zOutput.format(initialPosition.z));
  writeBlock("G43" + SP + "H0" + tool.number);
}
```

I ended here, as this is what is necessary to understand to get the basics.

This produced very basic G code to drill holes. Now we can add a simple drilling canned cycle to the post. To do this, we use onCyclepoint and onCycleEnd. To see how to output a very basic canned cycle, add this code to the end of your post file:

```
function onCyclePoint(x, y, z) {

  if (isFirstCyclePoint()) {
    repositionToCycleClearance(cycle, x, y, z);  // return to clearance plane and set absolute mode
    var F = cycle.feedrate;
   if  (cycleType == "drilling") {
       writeBlock(
       "G98 G90 G81" + xOutput.format(x)+ yOutput.format(y)+ zOutput.format(z) +
feedOutput.format(F));
     }
```

```
      } else {
    writeBlock(xOutput.format(x) + yOutput.format(y));
  }
}


function onCycleEnd() {
 if (!cycleExpanded) {
  writeBlock("G80");
  zOutput.reset();
 }
}
```

The onCycleEnd function will be output after the last hole in the CAM operation. The posted output for the operation Drill1 should look like this:

```
%
N6 T1 S500
N7 G0 X-0.85 Y-0.3154
N8 G0 Z+0.6
N9 G0
N10 G0
N11 G0 Z+0.2
N12 G98 G90 G81 Z-0.24  F5
N13 Y+0.35
N14 X+0.85
N15 Y-0.3154
N16 G80
N17 G0 Z+0.6
N18 G0 G91 G28
N19 G90
N20 M30
%
```

We should now look at simple indexing to another coordinate system. Change the onOpen section to look like this:

```
function onOpen() {
 var aAxis = createAxis({coordinate:0, table:true, axis:[1, 0, 0], range:[-360,360], cyclic:true,
preference:1});
 machineConfiguration = new MachineConfiguration(aAxis);
 setMachineConfiguration(machineConfiguration);
 optimizeMachineAngles2(1);

 writeln("%");
}
```

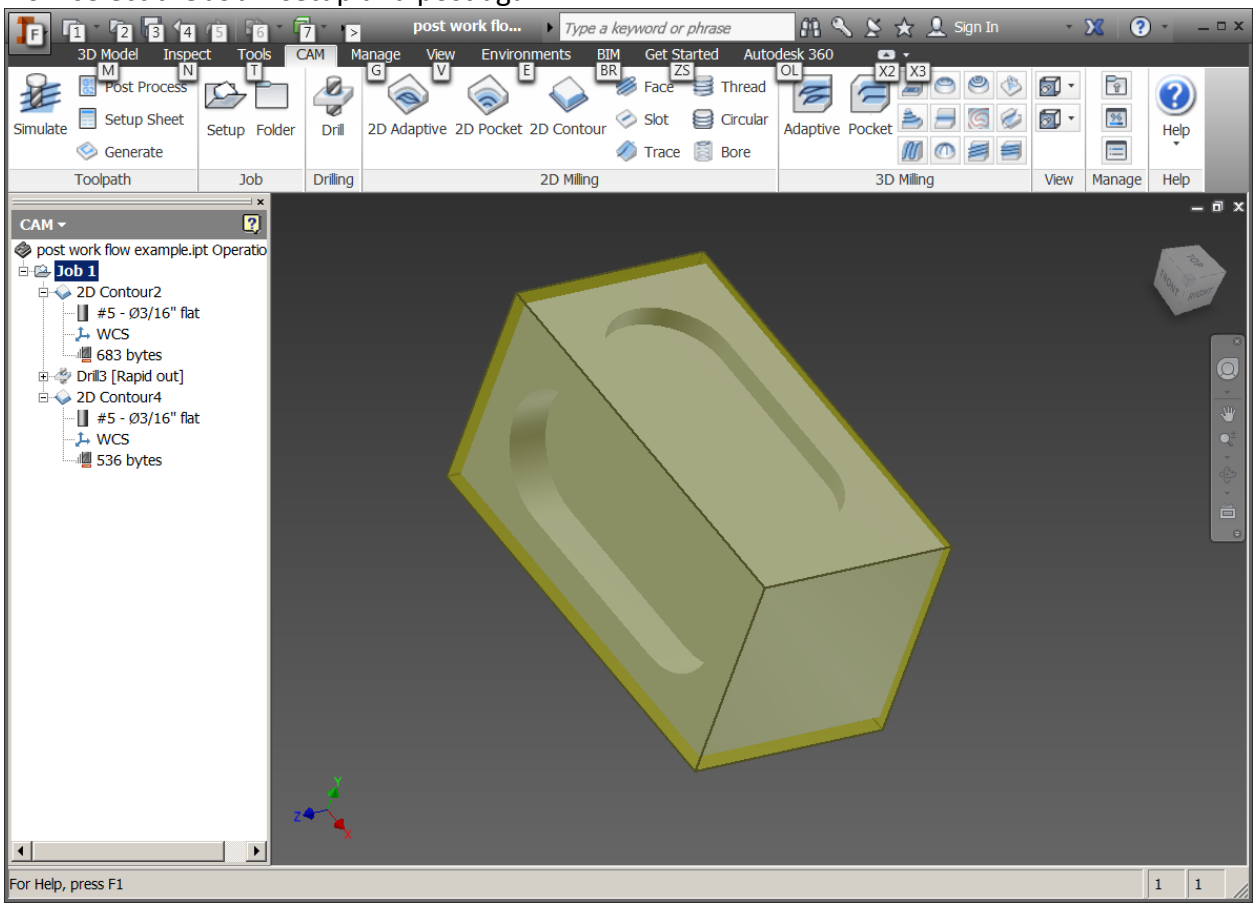
Also change the onSection in your post to look like this:

```
function onSection() {
  var abc = machineConfiguration.getABC(currentSection.workPlane);
  setRotation(machineConfiguration.getRemainingOrientation(abc, currentSection.workPlane));
  writeBlock("G0", "A" + abcFormat.format(abc.x));

  writeBlock("T" + tool.number, "S" + rpmFormat.format(tool.spindleRPM));

  xOutput.reset();
  yOutput.reset();
  zOutput.reset();
  var initialPosition = getFramePosition(currentSection.getInitialPosition());
  writeBlock("G0", xOutput.format(initialPosition.x), yOutput.format(initialPosition.y));
  writeBlock("G0", zOutput.format(initialPosition.z));
}
```

Now select the Job 1 setup and post again.



Now we see the A axis moves about the X Axis for the rotation.

```
%
N5 G0 A0
N6 T4 S5000
N7 G0 X-0.1 Y0.025
N8 G0 Z0.6
```

```
N9 G0
N10 G0 Z0.0394
N11 G1 Z-0.24 F20
N12 G1 X-0.2375 Y-0.075 G41
N13 G3 X0. Y-0.3125 I0.2375
N14 G1 X0.5 F40
N15 G3 Y0.3125 J0.3125
N16 G1 X-0.5
N17 G3 Y-0.3125 J-0.3125
N18 G1 X0.
N19 G3 X0.2375 Y-0.075 J0.2375
N20 G1 X0.1 Y0.025 G40
N21 G0 Z0.6
N22 G0 G91 G28
N23 G90
N24 G0 A0.
N25 T1 S500
N26 G0 X-0.85 Y-0.3154
N27 G0 Z0.6
N28 G0
N29 G0
N30 G0 Z0.2
N31 G0 Z0.16
N32 G1 Z-0.24 F5
N33 G0 Z0.2
N34 G0 Y0.35
N35 G0
N36 G0 Z0.16
N37 G1 Z-0.24 F5
N38 G0 Z0.2
N39 G0 X0.85
N40 G0
N41 G0 Z0.16
N42 G1 Z-0.24 F5
N43 G0 Z0.2
N44 G0 Y-0.3154
N45 G0
N46 G0 Z0.16
N47 G1 Z-0.24 F5
N48 G0 Z0.2
N49 G0 Z0.6
N50 G0 G91 G28
N51 G90
N52 G0 A-90
N53 T5 S5000
N54 G0 X0.4 Y0.3031
```

```
N55 G0 Z1.4525
N56 G0
N57 G0 Z0.8919
N58 G1 Z0.375 F20
N59 G1 X0.2969 Y0.2031 G41
N60 G3 X0.5 Y0. I0.2031
N61 G1 X-0.5 F40
N62 G3 X-0.2969 Y0.2031 J0.2031
N63 G1 X-0.4 Y0.3031 G40
N64 G0 Z1.4525
N65 G0 G91 G28
N66 G90
N67 M30
%
```

Now make this change to the post file [1, 0, 0] to [-1, 0, 0]:

```
var aAxis = createAxis({coordinate:0, table:true, axis:[-1, 0, 0], range:[-360,360], cyclic:true,
```

Post again and notice the A move became positive. This is how to change the direction of the rotary axis.

Future work:

Let's return to that program number. As we said before, NC controls have a limit on the name of the program for storage in the controls memory. We need to add some logic to limit the name.

```
function onOpen() {
  writeln("%");
if (programName) {
   var programId;
   try {
    programId = getAsInt(programName);
   } catch(e) {
    error(localize("Program name must be a number."));
    return;
   }
  writeln((programName ? (programName) : "") );
}
```

This will produce an error dialog and replace the NC program with an error file. This very simple post processor is intended for educational purposes only, and is not intended to be used for the manufacturing of parts in CNC machine tools.