

# *Post Building Techniques*

**Student Guide  
July 2007  
MT11060 — Post Builder 5.0**

# *Proprietary and restricted rights notice*

This software and related documentation are proprietary to UGS Corp.

Copyright 2007 UGS Corp. All Rights Reserved.

All trademarks belong to their respective holders.

# Contents

<b>The NX/Post Post Processor</b> .....	<b>1-1</b>
Post processing .....	1-2
Summary .....	1-6
<b>Build a Post Processor with Post Builder</b> .....	<b>2-1</b>
Post Builder overview .....	2-3
Activity — Preparation for using the Post Builder .....	2-4
Post Builder menu and tool bars .....	2-6
Use the Post Builder to create a new Post Processor .....	2-9
Activity — Introduction to Post Builder .....	2-11
NX/Post Builder Parameters .....	2-15
Activity — Machine Tool Parameter selections .....	2-17
Program and Tool Path Property page .....	2-19
Activity — Word summary .....	2-21
Program & Tool Path Property page (continuation) .....	2-27
About the interface .....	2-28
Activity — Program & Tool Path parameters .....	2-34
NC Data Definitions property page .....	2-41
Activity — NC Data Definition .....	2-44
Create new M or G code groups .....	2-53
Activity — Create a new M-Code group .....	2-54
Output Settings property page .....	2-56
Activity — Output settings .....	2-60
Post Files Preview property page .....	2-62
Summary .....	2-63
<b>Post Builder for Wire EDM applications</b> .....	<b>3-1</b>
Use Post Builder to create 2-axis and 4-axis Wire EDM post processors ..	3-2
Activity — Create a 2–axis Wire EDM post .....	3-5
Summary .....	3-8
<b>Post Builder for 5-Axis mill applications</b> .....	<b>4-1</b>
Use Post Builder to create 5-axis Mill post processors .....	4-2
Activity – Create a 5–Axis Mill Post with Post Builder .....	4-8
Summary .....	4-11

<b>Post Builder for lathe applications</b>	<b>5-1</b>
Use Post Builder to create Lathe post processors	5-2
Activity — Create a Lathe Post with Post Builder	5-5
Summary	5-7
 <b>Create Mill-Turn post processors</b>	 <b>6-1</b>
Mill-Turn centers	6-2
Mill-Turn and the Post Builder	6-3
Heads for Mill-turn centers	6-4
Create a Mill-Turn Post Processor	6-5
5 Axis Mill-Turn centers	6-10
5-Axis Mill-Turns and the Post Builder	6-12
Activity — Create a 5–Axis multi-link mill-turn post	6-13
Summary	6-24
 <b>Tcl basics for Post Builder</b>	 <b>7-1</b>
Tcl	7-2
Tcl command structure	7-3
Tcl scripts	7-4
Tcl control of word structure	7-5
Tcl variables	7-6
Tcl mathematical expressions	7-7
Variable examples	7-8
Tcl variable definitions	7-9
Variable substitution	7-10
Variable substitution examples	7-11
Activity — Tcl basics	7-12
Tcl procedures and functions	7-14
Tcl I/O	7-16
Tcl special characters	7-17
Construct a simple procedure	7-18
Tcl flow control structures	7-21
Activity — Tcl flow constructs	7-24
Tcl formats	7-27
Tcl and Unigraphics	7-28
Tcl reference manuals	7-29
Summary	7-30
 <b>Customize a post processor with Post Builder</b>	 <b>8-1</b>
Activity — Custom commands	8-8
Summary	8-18
 <b>User Defined Events (UDEs)</b>	 <b>9-1</b>
User Defined Events	9-2

User Defined Event syntax .....	9-3
Activity — Create user defined events .....	9-6
Activity — Modify the coolant UDE for thru-spindle .....	9-11
Summary .....	9-15
<b>Virtual NC Controller .....</b>	<b>10-1</b>
Integrated Simulation and Verification overview .....	10-2
Machine Tool Driver .....	10-3
Activity — Use Post Builder to create a VNC .....	10-5
Summary .....	10-7
<b>A guide to best practices of building a Post Processor .....</b>	<b>11-1</b>
<b>Custom command examples .....</b>	<b>A-1</b>
<b>Advanced Post Building topics .....</b>	<b>B-1</b>
Guide to modifying and customizing existing post processors .....	B-3
Activity — Modify an event handler .....	B-5
Definition file .....	B-7
Activity — Modify a definition file .....	B-8
Machine Kinematics .....	B-12
Circles .....	B-13
Advanced Kinematics .....	B-14
Rotary axes designations .....	B-15
4th axis center offsets .....	B-16
5th axis center offsets .....	B-17
Axis rotation (standard or reverse) .....	B-18
Zero position offset .....	B-21
Pivot point .....	B-23
Dual table kinematics .....	B-25
Special case 5-axis dual table .....	B-27
UG/Post Postprocessing using Runugpost .....	B-29
Summary .....	B-30
<b>Index .....</b>	<b>Index-1</b>



## Lesson

# 1 *The NX/Post Post Processor*

### Purpose

This lesson describes the post processing function of NX and the various components of a *NX/Post* post processor. An overview of the *Post Builder* module is also presented.

### Objective

Upon completion of this lesson, you will be able to:

- Understand the terminology and interrelationships of the various components of *NX/Post*.
- Understand the use of the *Post Builder* as a tool to build post processors.

## Post processing

You use the Manufacturing applications of NX to generate tool paths. The tool path consists of GOTO points and other information that controls the movement of the tool. This tool path usually cannot be used for machining since each machine tool/controller combination has different requirements, tool change requirements, and software characteristics, such as the sequence of G codes which are permitted on a line of output.

The tool path must be formatted to match the unique characteristics of the machine tool/controller combination. The procedure of modifying this generic tool path to a form that can be understood and used by the machine tool controller is called post processing.

Two elements are required for post processing. They are:

- Tool path - A NX internal tool path.
- Post processor - this is a computer program that reads, converts and reformats tool path information for a particular machine tool/controller combination.

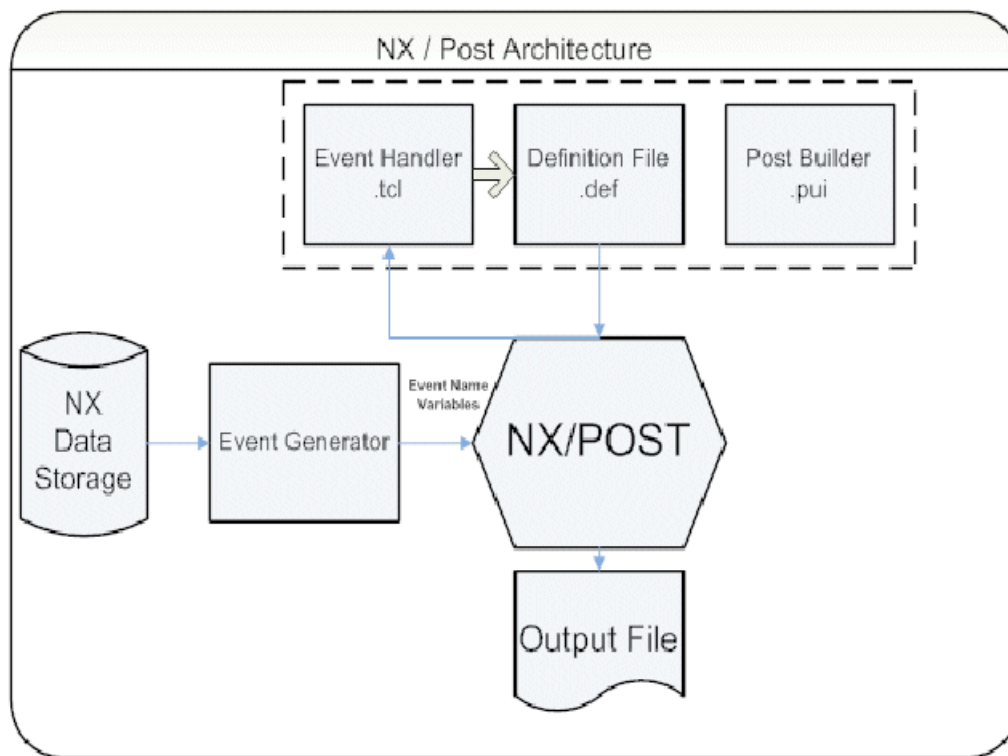
### NX/Post execute

NX provides a post processor, *NX/Post*, which utilizes NX tool path data as input, and outputs machine controller readable NC/CNC code. Post Processors for *NX/Post* are customizable through the use of user created Event Handler and Definition files. These files, in conjunction with *NX/Post*, are used to generate output for the simplest to the very complex of machine tool/controller combinations.

The *NX/Post* processor can be used to generate output for simple milling machines and lathes to ultra complex multi-axis (4+ axis) machining and production centers (a production center is considered to be a milling/turning type machine). The extensibility of *NX/Post* post processors is achieved through the scripting language *Tcl* and the use of the NX concept of Definition files.

The following flowchart illustrates the steps required to process (post process) tool path data in an acceptable format for a machine tool/controller using the *NX/Post* post processor:





The *NX/Post* execute module consists of the following components:

- *Event Generator* - sends Events and variables to *NX/Post* when you post process. An Event is a collection of data which is processed by *NX/Post*, creating data which causes a specific action(s) by the machine tool/controller.
- *Event Handler* - is a file containing a specific set of instructions, written in the Tcl scripting language, dictating how each event type is to be processed. This file is created when using Post Builder.
- *Definition file* - is a file containing specific information about machine tool/controller format. This file is also created when using Post Builder.
- *Output file* - is a machine tool readable file generate by *NX/Post*, passed to the machine tool/controller, that executes specific machine tool instructions.

The Event Generator, Event Handler, and the Definition file are interdependent and together convert the internal tool path into a set of instructions that can be read and executed by the specific machine tool/controller combination.

## Manufacturing Output Manager (MOM)

The Manufacturing Output Manager, referred to as MOM, is a utility program used by *NX/Post* for generating output based upon data that is stored within the internal tool path. *NX/Post* uses the MOM to start, add data and specify functions to the interpreter, and to load Event Handlers and Definition files.

## Post Builder

The *NX/Post* mechanism uses Tcl (Tool Command Language) scripts and files for post processing. These files extract information from the part file, process this information according to defined rules, formats the information for output and then outputs the data to a file which is later used by the machine control for machining a part. These files are highly customizable, and if manually edited, require the working knowledge of the Tcl scripting language.

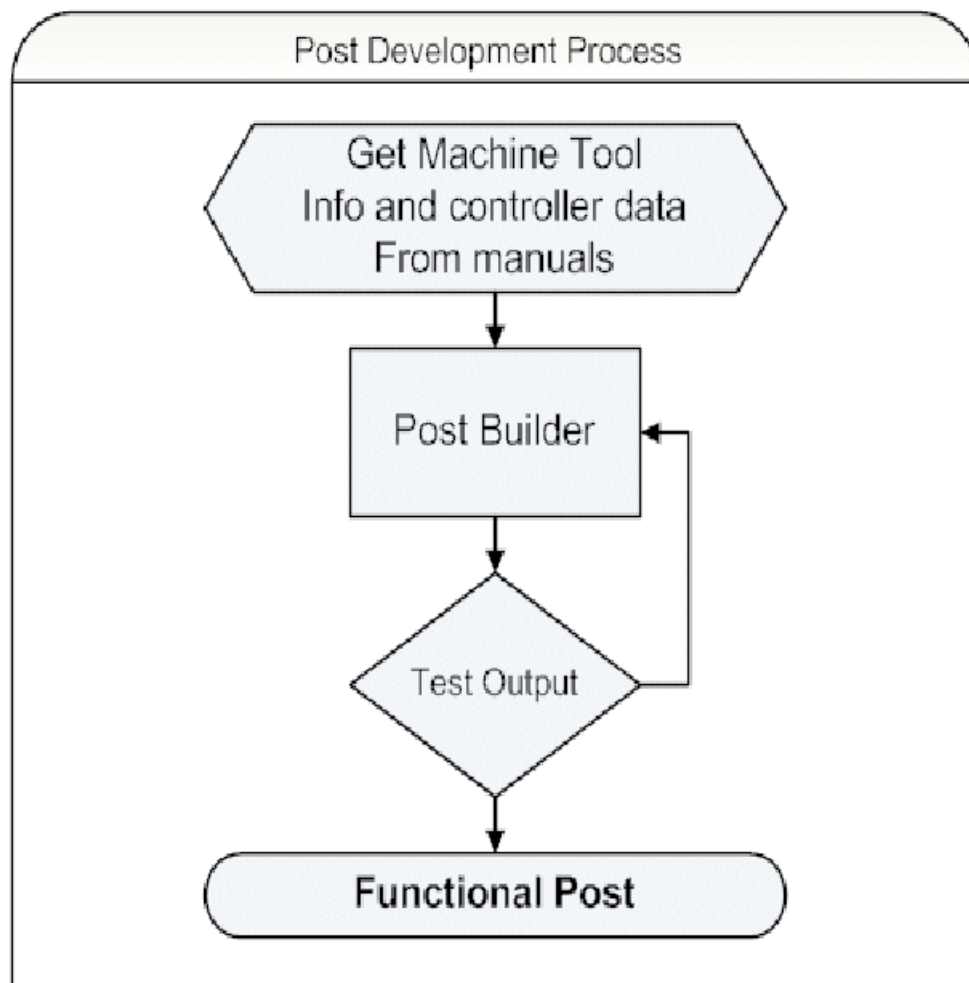
*Post Builder* provides a graphical User Interface for building posts.

*Post Builder* is very flexible and allows for the definition of various types of output blocks and word addresses. Sequence of output in the NC output file is very easy to control for blocks involving the start of program, start of operation, end of operation, end of program, tool changes and canned cycles.

*Post Builder* currently configures post processors for the following:

- 3-axis milling machines
- 3-axis mill-turn centers (XZC)
- multi-axis mill-turn centers
- 4-axis milling machines with a rotary table or a rotary head
- 5-axis milling machines with dual rotary heads or rotary tables
- 5-axis milling machines with rotary head and rotary table
- 2-axis lathes
- 2 and 4-axis Wire EDM machines

The following flowchart illustrates the process of building a post processor using the *Post Builder*:



# 1

## Summary

In this lesson you were introduced to:

The post processing function of *NX/Post*.

Interrelationships and functionality of the various *NX/Post* components.

## Lesson

# 2 *Build a Post Processor with Post Builder*

2

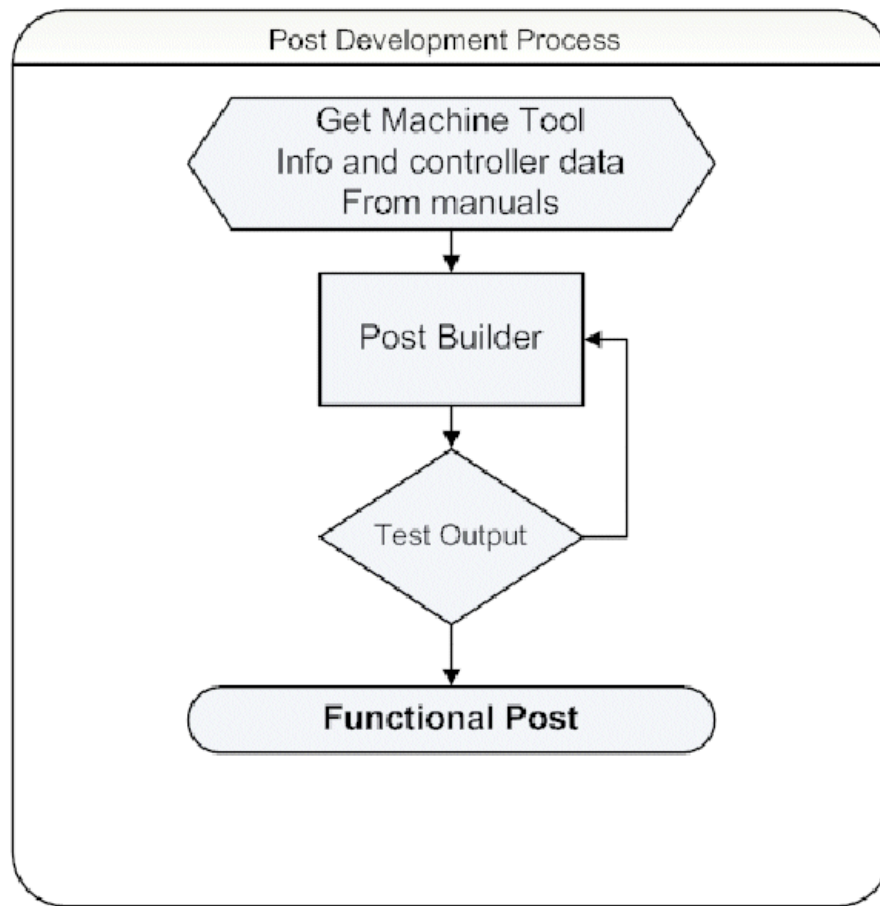
### Purpose

This lesson describes the procedures of building post processors through the use of the **Post Builder**.

### Objective

Upon completion of this lesson, you will be able to:

- Understand the various components of **Post Builder**.
- Use the **Post Builder** to build a 3-axis mill post processor.



## Post Builder overview

**Post Builder** provides an easy to use interface for creating and modifying post processors. Drag and drop functions allow the creation of formats, addresses, sequences and blocks. The **Post Builder** creates the definition of output blocks, formats for addresses and controls the sequence of output for the start and end of program, start and end of operation and tool path Events.

The **Post Builder** works in conjunction with three files.

- The **Definition** file, with .def extension, contains information about the machine tool/controller functionality and format requirements.
- The **Event Handler**, with .tcl extension, contains instructions of how each Event type is to be processed.
- The **Post Builder parameter** file, with .pui extension, contains the parameters that are used by **Post Builder** for an individual post. This file is referenced whenever you would edit or customize a post processor using the **Post Builder**.

An existing library of post processors is provided by UGS at [http://ftp.ugs.com/unigraphics/pb\\_posts/post\\_index.htm](http://ftp.ugs.com/unigraphics/pb_posts/post_index.htm).

Machine tools and machine controllers in industry vary greatly, you should be aware that postprocessors made available on this ftp site should be considered as example postprocessors only and considerable adjustment of these postprocessors may be required to suit the specific machine tool.

The activities, in this lesson, will use a copy of the main CAM mach/resource directory. The logic behind this concept is that you can experiment with various ideas and techniques without the worry of corrupting any system files that would normally be used by **NX/Post** or **Post Builder**.

The following activity will take you through the process of creating a copy of the mach/resource directory structure to your home directory.

## Activity — Preparation for using the Post Builder

In this activity you will make a copy of the MACH/RESOURCE directory structure in the home directory and modify the directories for read/write access.

**Step 1:** Copy the mach/resource directory.

- ☐ Start NX.
- ☐ On the NX Main Menu Bar, click **Help**→ **NX Log File** to verify the **mach/resource** directory being used. Search for the environment variable **UGII\_CAM\_RESOURCE\_DIR**.
- ☐ Open an XP Explorer window and locate the directory from the previous action item.
- ☐ Highlight the **mach/resource** directory, right click on **mach/resource** directory and select **Copy**.
- ☐ Highlight your home directory, right click on your home directory and select **Paste**.

**Step 2:** Copy the environment file, ugi\_env.dat to your home directory.

- ☐ From the XP Explorer window locate the **ugii\_env.dat** file in the **/NXxx0/UGII** directory (note: xx represents current version).
- ☐ Highlight the **ugii\_env.dat** file, right click on the **ugii\_env.dat** file and select **Copy**.
- ☐ Highlight your home directory, right click on your home directory and select **Paste**.

**Step 3:** Edit the ugi\_env.dat file to redefine your mach/resource directory location.

- ☐ Highlight the **ugii\_env.dat** file from your home directory, right click on the **ugii\_env.dat** file and select **Open with Wordpad** (note: prior to editing this file, you may have to associate the file with either Wordpad or Notepad).
- ☐ Scroll down the file until you find the following line:  
**UGII\_CAM\_RESOURCE\_DIR=\${UGII\_CAM\_BASE\_DIR}resource/**  
 and change the line to  
**UGII\_CAM\_RESOURCE\_DIR=\${HOMEDRIVE}\${HOMEPATH}resource/**  
 where **HOMEDRIVE** is the letter designator of the disk drive where your home directory is located and **HOMEPATH** is the directory structure.



- ☐ **Save** the file and exit from the editor.

**Step 4:** Change protection on the resource directory that was created.

- ☐ Navigate to your home directory, highlight the resource directory, right click, select properties, select the security tab, highlight the user name **Everyone** in the upper dialog, on the lower dialog, check on **Full Control**.
- ☐ Choose **OK** until you return to the main Explorer dialog.

**Step 5:** Restart NX.

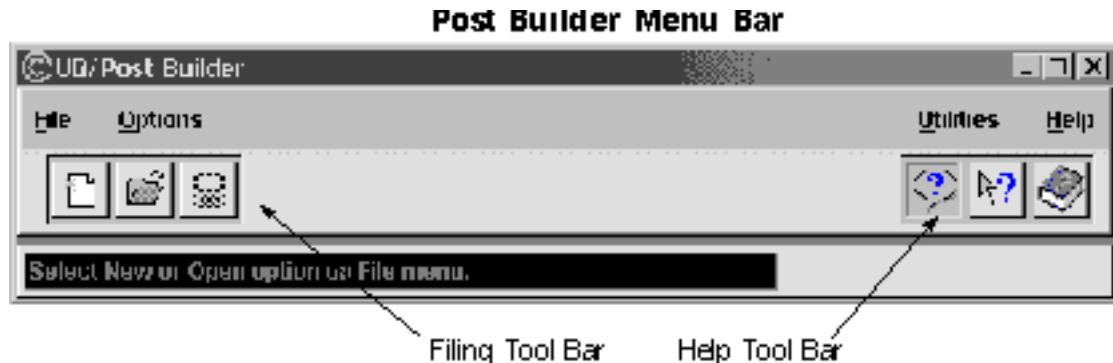
- ☐ Log off NX and then log back on.
- ☐ On the NX Main Menu bar, select **Help** → **NX Log File** to verify the updated **/mach/resource** directory is being used.

This completes this activity.

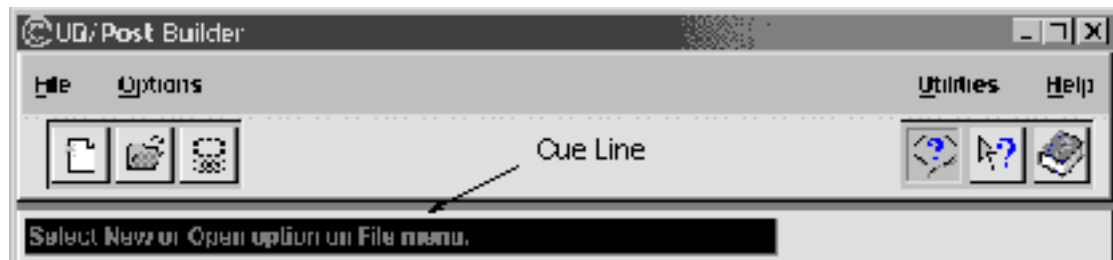
## Post Builder menu and tool bars

**Post Builder** is started by choosing from the desktop menu bar, **Start**→**All Programs**→**NX**→**Post Tools**→**Post Builder**

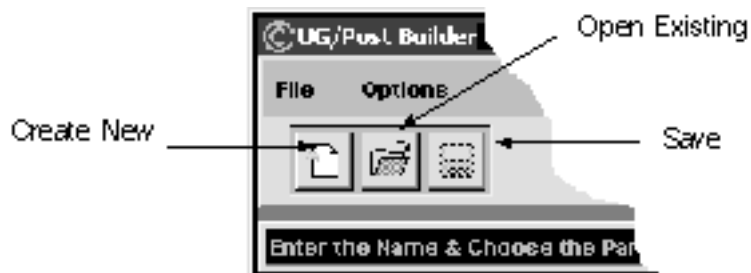
When you initially enter the **Post Builder** you will see the **Post Builder Menu Bar**. The Menu Bar contains a **Cue Line** and two **Tool Bars**.



The **Cue line** displays prompt messages on expected input by the current option. These messages indicate the next action which you need to take.



The **Post Filing** toolbar contains three command buttons associated with creating **new**, **opening** and **saving** post processors.

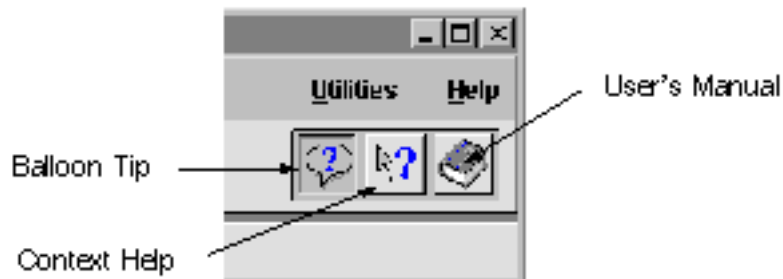


**Create New** option allows for creation of a new post processor.

**Open Existing** option allows for modification or editing of an existing post processor.

**Save** option allows for saving the post processor currently being worked upon.

The **Help tool bar** contains command buttons associated with help options.

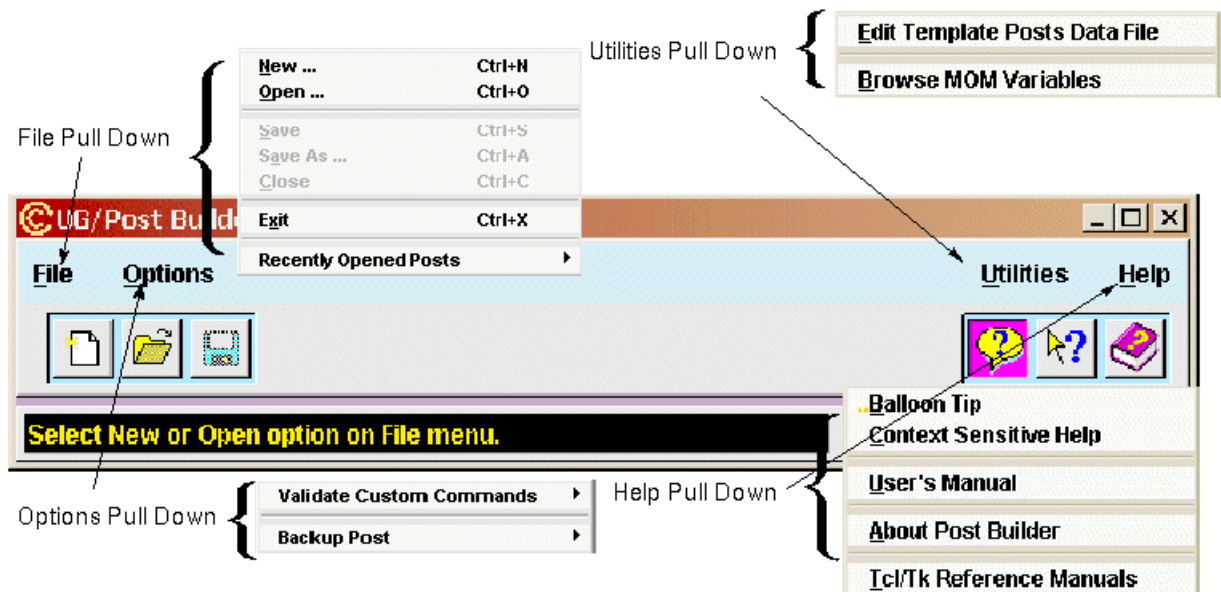


**Balloon Tip** option, when turned on, displays a message balloon that contains descriptions concerning the functionality of the widget, whenever the cursor stops at a widget on the dialog.

**Context Help** option when turned on, changes the pointer into a question mark. You can then click on an icon or item on the display that you are in question about and will see an explanation of the function of the icon or item. Click on the explanation window to close.

**User's Manual** is a complete online manual on the **Post Builder**. The **User's Manual** also contains MOM events and variables.

Pull Down menus are also available for **File**, **Options**, **Utilities** and **Help**.



The **File** list allows for the creation, opening, saving and closing of post processor files. It also allows you to access a list of the most recently opened posts as well as exiting from the **Post Builder** program.

The **Options** list allows for the validation of the syntax, addresses, blocks and formats used in Custom Commands as well as the number of backup files created.

The **Utilities** list allows for modification of the **Template\_post.dat** data file and the addition of MOM variables.

The **Help** list allows for the activation of **Balloon Tips**, **Context Sensitive Help**, access to the **User's Manual**, **Release Notes** and specific information concerning the version of the **Post Builder** being utilized.

## Use the Post Builder to create a new Post Processor

To create a new post processor, the **New** option (or Create New command button from the Post Filing Tool Bar) from the **File** pull down menu is selected. The **Create New Post Processor** menu is displayed which allows for post processor naming and description, output units and machine tool and controller type.

**Create New Post Processor**

Post Name: new\_post

Description: This is a 3-Axis Milling Machine.

**Post Output Unit**

☒ Inches ☐ Millimeters

**Machine Tool**

☒ Mill  
☐ Lathe  
☐ Wire EDM  
3-Axis

**Controller**

☒ Generic ☐ Library ☐ User's

3-Axis  
3-Axis Mill-Turn (XZC)  
4-Axis with Rotary Table  
4-Axis with Rotary Head  
5-Axis with Dual Rotary Heads  
5-Axis with Dual Rotary Tables  
5-Axis with Rotary Head and Table

OK Cancel

The **Post Name** field is the name of the post processor being created. Spaces are not allowed.

The **Description** field allows for text describing machine tool/controller characteristics and any other miscellaneous information.

**Post Output Units** selection buttons allow for inch or metric output.

**Machine Tool** allows the selection of various milling (multi-axis) machine, lathe configurations and Wire EDM type machines.

**Controller** allows the selection of **generic**, **library** or **User's** machine controllers.

- **Generic** controller contains defaults for a generic control
- **Library** allows the selection of controller from a NX supplied list
- **User's** allows the selection of post processors by browsing for specific post processors

NOTE: You can select a previously created post processor.

The **Machine Tool** and **Controller** selections determine base files used to create the post processor which contains various Events, commands and procedures.

The **Ok** button enables acceptance of specifications for the machine tool and controller and advances you to the main editor portion of the user interface.

The **Cancel** button will dismiss the current menu, terminates the process of creating a new post processor and returns you to the **Post Builder** menu bar.

When you create a new post processor, you can add the name of the post to the **template\_post.dat** file. This file contains a list of post processors for NX/Post to use. It defines a list of post processors that are available in the Post Process dialog. The name of the post processor can be added by using the **Utilities** pull down menu from the main tool bar.

The following activity will introduce you to the basic concepts and use of the **Post Builder** module.

## Activity — Introduction to Post Builder

**Step 1:** Start the **Post Builder**.

- ☐ On the menu bar, choose **Start→All Programs→Post Tools→Post Builder**
- ☐ Select **New** from the **File** menu and in the **Post Name** field, name the post processor **\*\*\*\_my\_post**, where **\*\*\*** stands for your initials.

**Note:** Use lower case characters only and no spaces.

**Step 2:** Select various initial options.

For **Post Output Unit** you will accept the default of **inches**.

For **Machine Tool** you will accept the default of **Mill**.

Notice the 3-axis button bar.

- ☐ Select the 3-axis button bar and notice the choices available.

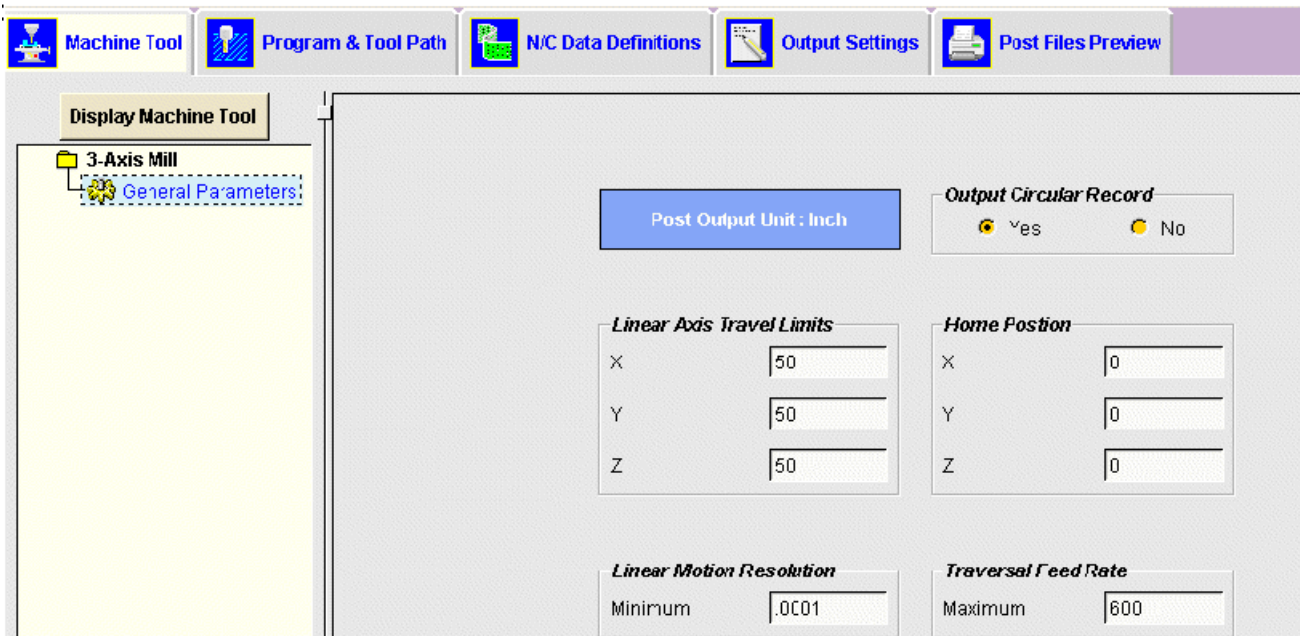
You will accept the default for 3-axis.

- ☐ Dismiss the **3-axis**.

For **Controller** you will accept the default of **Generic**. Some common controllers are already provided. If you were to select the **Library** option, a list of available controllers would be available for selection as a template in building your post.

- ☐ Click **OK**.

The **Main Editor** property page (tab) is displayed.



**Step 3:** Save your newly created post processor to your home directory.

- ☐ Select **File** from the menu bar and then select **Save As**.
- ☐ Filter to your home post processor directory.
- ☐ Visually check the **File name** box, if the file name shown is the same as the file name of your post processor, select **OK**, otherwise type in the correct name and then select **OK**.

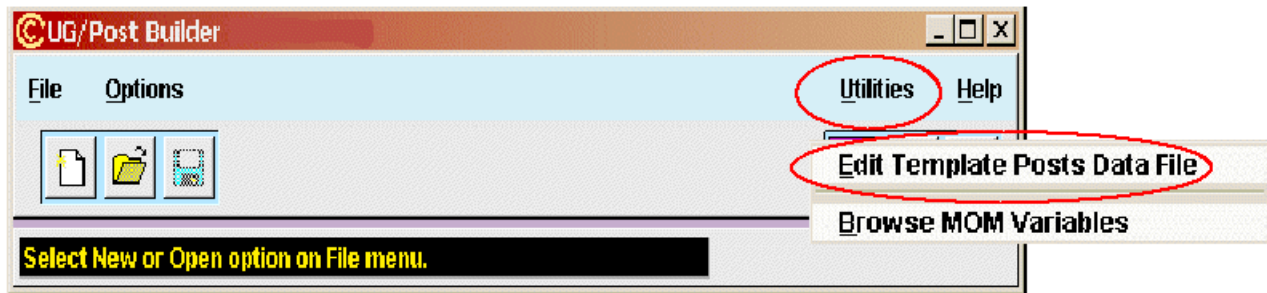
When you save your file, three files are actually saved. The files have the extension **.pui**, **.tcl**, and **.def**.

In order for your newly created post processor to be included in the NX Post Processing window by default, you must save the post to your home post processing directory and modify the **template\_post.dat** file to include the new post processor. This file is also located in the home post processor directory. This file is added to the template\_post.dat file by using the **Utilities** function from the main menu bar.

**Step 4:** Add your newly created post processor to the template\_post.dat file.



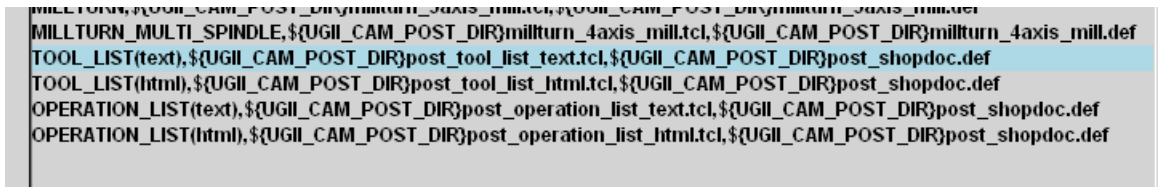
- ☐ Select **Edit Template Posts Data File** from the **Utilities** list.



The Install Posts dialog is displayed.

- ☐ Browse to your post processor directory, then select the `template_post.dat` file and choose **OK**.

Click the **TOOL LIST** line as shown below, then select the **NEW** button.



The Open dialog is displayed.

- ☐ Navigate to your home directory and select the post processor which you previously saved.
- ☐ Click **Open**.

The path and name of the file are inserted into the **Template\_post.dat** file.

- ☐ Click **OK**.

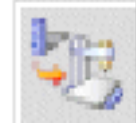
The **Save As** dialog appears.

- ☐ If necessary, select the file name **template\_post.dat**.
- ☐ Click **Save**.
- ☐ Click **Yes** to the **Save As** warning message to replace the `template_post.dat` file.

**Step 5:** Verify the post processor has been added to the Post Process dialog.

- ☐ Return to a NX session and retrieve the part file **mill\_test** from your **student\_home/parts** directory.

- ☐ Verify the post processor by entering the Manufacturing



Application in NX and select **Post Processing**

You will see your post processor, **\*\*\*\_my\_post**, under Available Machines on the Post Process dialog.

- ☐ Click **Cancel**.
- ☐ Return to the **Post Builder** main dialog.

This concludes the activity.

## NX/Post Builder Parameters

As you proceed to build a post processor using the **Post Builder**, the user interface takes on the appearance of a tabbed notebook. There are five major tabs (also referred to as property pages) which are displayed.

- **Machine Tool**
- **Program & Tool Path**
- **NC Data Definitions**
- **Output Settings**
- **Post Files Review**

When a major property page is selected, menus and or sub property pages are displayed which allows for the setting of various parameters used in creation of your post processor.

## Machine Tool Property Page

2

The screenshot shows the 'Machine Tool Property Page' with the 'Machine Tool' tab selected. The left pane displays a tree view under '3-Axis Mill' with 'General Parameters' selected. The main area contains the following sections:

- Post Output Unit :** Inch
- Output Circular Record:** Yes (selected), No
- Linear Axis Travel Limits:**
  - X: 50
  - Y: 50
  - Z: 50
- Home Position:**
  - X: 0
  - Y: 0
  - Z: 0
- Linear Motion Resolution:** Minimum: .0001
- Traversal Feed Rate:** Maximum: 600
- Axis Multipliers:**
  - Diameter Programming:** 2X (selected), 2Y (selected)
  - Mirror Output:** -X (selected), -Y (selected), -Z (selected), -I (selected), -J (selected), -K (selected)
- Initial Spindle Axis:**
  - I: 0
  - J: 0
  - K:

The **Machine Tool** property page allows the selection of output to include Circular Records or to output linear motion only. You can also specify **Linear Axis Travel Limits**, **Home Position**, **Linear Motion Step Size** and **Traversal Feed Rate**, **Axis Multipliers** and vectors representing the **Initial Spindle Axis**. The **Display Machine Tool** button gives a simple, generic, representation of the motions of the machine tool being configured.

The **Default** button returns you to the state when the post was last saved for the current "screen".

The **Restore** button returns you to the state when you entered the current "screen".

## Activity — Machine Tool Parameter selections

In this activity you will examine and modify some of the various Machine Tool Parameters located within the Machine Tool selection of the Post Builder and will then restore them to their original state.

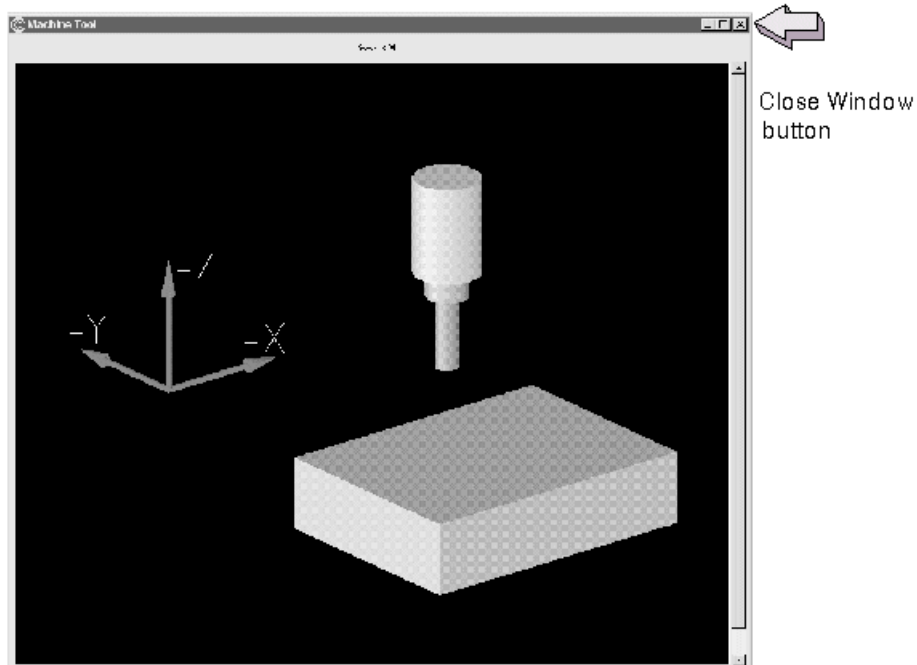
**Step 1:** Examine the various Machine Tool Parameters.

- ☐ Click **Display Machine Tool**.



2

Notice the display of a simple 3-axis mill.



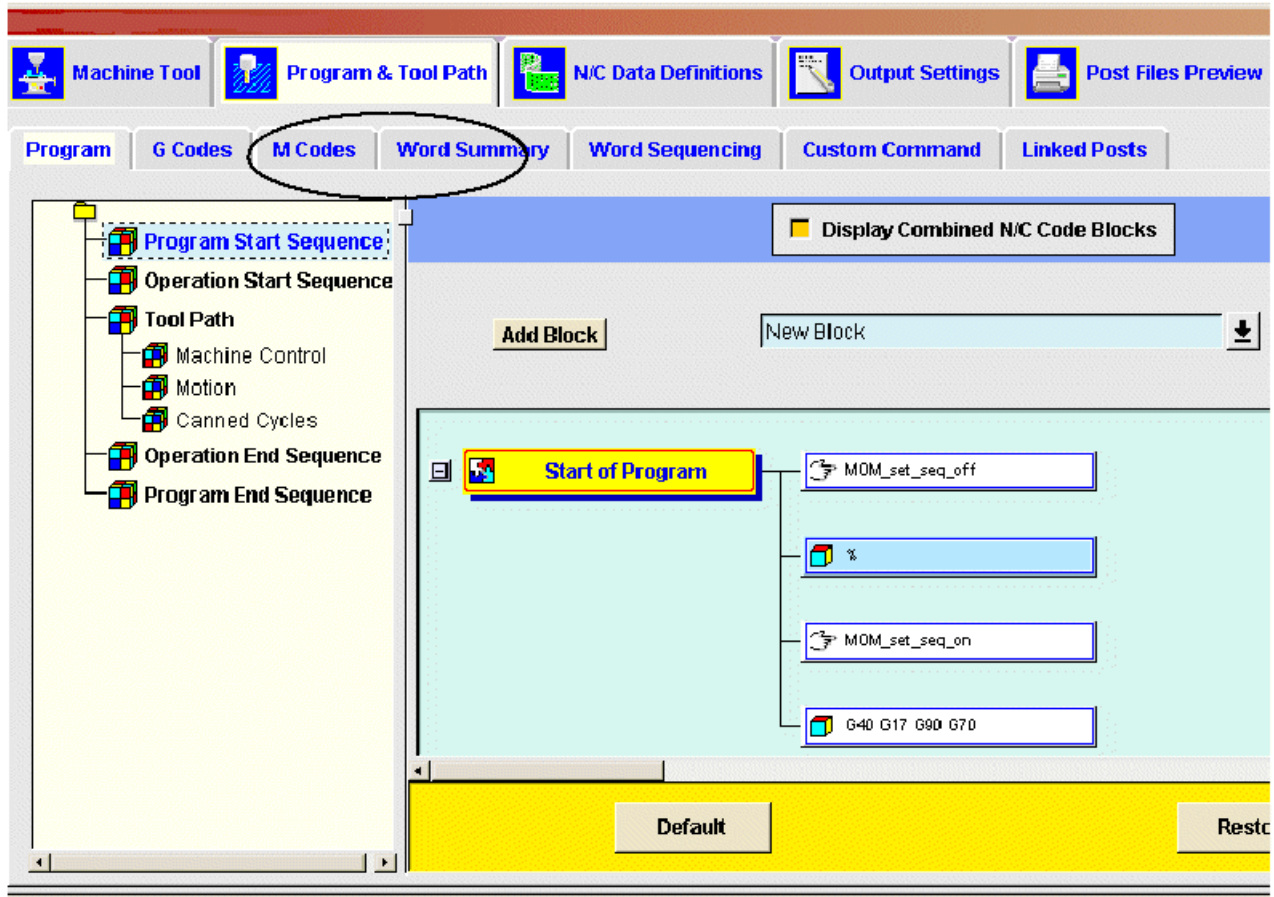
- ☐ Click **Close** on the Machine Tool title bar.
- ☐ Set the linear travel limits to the following: **X=60, Y=50** and **Z=40**.
- ☐ Select the **Restore** button.



Notice that the X, Y and Z axis limits are returned to their original settings.

This concludes the activity.

## Program and Tool Path Property page



The **Program and Tool Path** property page allows for creation, modification and customization of the Event Handlers for all Events which occur by the use of multiple function property pages. These functions, presented in the tab format, are:

**Program** allows the addition, modification and customization of Program Start Sequences, Operation Start Sequences, Tool Path Events including Machine Control Events, Motion Events, Canned Cycles Events, Operation End Sequences and Program End Sequences.

**G Codes** property page will allow you to specify default G-codes used throughout the post processor. Changing any of the G-codes in this list will update the G-codes globally.

**M Codes** property page will allow you to specify default M-codes used throughout the post processor. Changing any of the M codes in this list, will update the M-codes globally.

**Word Summary** property page provides a summary of all addresses and allows the individual modification of the various elements used in data definition of the address. This property page allows the modification of groups of words that use the same format and also allows modification of the format itself. If

you want to specify a different format for only one of the words, you would go to the Format section of NC Data Definitions property page to make the change. The following parameters are available in Word Summary:

- Word is the same that you will find in the word section of the NC Data Definitions section.
- Leader/Code - allows you to modify the leader for any word. The leader is the character that precedes the numeric information. For example, if the word is X20.0000, then the leader is X. You can either enter a new leader or right click on the existing leader and choose from the list.
- Data Type - can be either numeric or text. Specify text when the code needed cannot be formatted from a number directly into a word address and a value. If, for example, you need a G84.1 for tapping and G81 (without a decimal) for drilling, you would make the G-motion word text and change all of the G codes to G81 instead of 81 with a leader of G. You may also go to the format property page in NC Data Definitions to change only the G-motion word to text.
- Plus (+) - allows a plus sign for all positive numbers. If set to **no** the (+) sign is not generated. A minus sign (-) is generated for all numeric data if the value is negative.,
- Lead Zero - will allow the output of leading zeros.
- Integer - controls number of digits output to the left of the decimal point. An error is generated if the numeric data exceeds the format specified.
- Decimal (.) - controls whether or not numeric data has a decimal point. An error is generated if the decimal is suppressed and leading or trailing zeroes are not output.
- Fraction - determines how many digits are output to the right of the decimal point.
- Trail Zero - will allow the output of trailing zeros.

To change how the post processor will output a value of zero for coordinate data, select either the **zero\_int** or **zero\_real** option for formats in the NC Data Definitions section under the Format property page. This will output a zero in front of the decimal point.



## Activity — Word summary

In this activity you will modify some of the various word formats and change the sequence number. You will find these items in the Program and Tool Path and Word Summary section of the Post Builder.

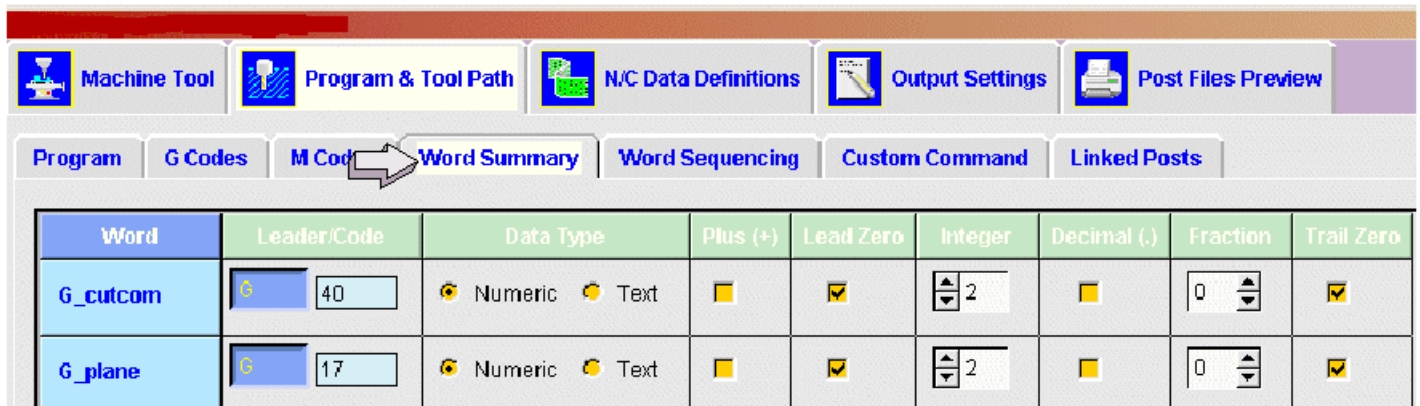
2

**Step 1:** Enter the Word Summary subsection of the Program and Tool Path section of **Post Builder**.

- ☐ Select the **Program & Tool Path** property page from the **Post Builder** selection menu.



- ☐ Select **Word Summary** property page.



**Step 2:** Change the X, Y, and Z-axis word format to 3.3.

- ☐ Scroll down the **Word** column until you reach the **X** address.
- ☐ In the row of the **X** address, locate the **integer** column and change the value from 4 to 3.

Word	Leader/Code	Data Type	Plus (+)	Lead Zero	Integer	Decimal (.)	Fraction	Trail Zero
G	G 01	Numeric Text	<input type="checkbox"/>	<input checked="" type="checkbox"/>	2	<input type="checkbox"/>	0	<input checked="" type="checkbox"/>
X	X 1.234	Numeric Text	<input type="checkbox"/>	<input type="checkbox"/>	4	<input checked="" type="checkbox"/>	4	<input type="checkbox"/>
Y	Y 1.234	Numeric Text	<input type="checkbox"/>	<input type="checkbox"/>	4	<input checked="" type="checkbox"/>	4	<input type="checkbox"/>

2

- ☐ In the row of the **X** address, locate the **fraction** column and change the value from **4** to **3**.

Word	Leader/Code	Data Type	Plus (+)	Lead Zero	Integer	Decimal (.)	Fraction	Trail Zero
G	G 01	<input checked="" type="radio"/> Numeric <input type="radio"/> Text	<input type="checkbox"/>	<input checked="" type="checkbox"/>	2	<input type="checkbox"/>	0	<input checked="" type="checkbox"/>
X	X 1.234	<input checked="" type="radio"/> Numeric <input type="radio"/> Text	<input type="checkbox"/>	<input type="checkbox"/>	4	<input checked="" type="checkbox"/>	4	<input type="checkbox"/>
Y	Y 1.234	<input checked="" type="radio"/> Numeric <input type="radio"/> Text	<input type="checkbox"/>	<input type="checkbox"/>	4	<input checked="" type="checkbox"/>	4	<input type="checkbox"/>

Notice that changing the X address word format also changes all other axis word formats.

**Step 3:** Change the F address format to 3.1.

- ☐ Scroll down the **Word** column until you reach the **F** address.
- ☐ In the row of the **F** address, locate the **integer** column and change the value from **7** to **3**. The **fraction** column is set to **2**, change the value to **1**.

Word	Leader/Code	Data Type	Plus (+)	Lead Zero	Integer	Decimal (.)	Fraction	Trail Zero
F	F 1.23	<input checked="" type="radio"/> Numeric <input type="radio"/> Text	<input type="checkbox"/>	<input type="checkbox"/>	7	<input checked="" type="checkbox"/>	2	<input type="checkbox"/>
S	S 1	<input checked="" type="radio"/> Numeric <input type="radio"/> Text	<input type="checkbox"/>	<input type="checkbox"/>	5	<input type="checkbox"/>	0	<input checked="" type="checkbox"/>
T	T 01	<input checked="" type="radio"/> Numeric <input type="radio"/> Text	<input type="checkbox"/>	<input checked="" type="checkbox"/>	2	<input type="checkbox"/>	0	<input checked="" type="checkbox"/>

**Step 4:** Change the S address format to 4.

- ☐ In the row of the **S** address, locate the **integer** column and change the value from **5** to **4**. The **fraction** column is set to 0 and does not need to be changed.

Word	Leader/Code	Data Type	Plus (+)	Lead Zero	Integer	Decimal (.)	Fraction	Trail Zero
F	F 1.23	<input checked="" type="radio"/> Numeric <input type="radio"/> Text	<input type="checkbox"/>	<input type="checkbox"/>	7	<input checked="" type="checkbox"/>	2	<input type="checkbox"/>
S	S 1	<input checked="" type="radio"/> Numeric <input type="radio"/> Text	<input type="checkbox"/>	<input type="checkbox"/>	5	<input type="checkbox"/>	0	<input checked="" type="checkbox"/>
T	T 01	<input checked="" type="radio"/> Numeric <input type="radio"/> Text	<input type="checkbox"/>	<input checked="" type="checkbox"/>	2	<input type="checkbox"/>	0	<input checked="" type="checkbox"/>

**Step 5:** Allow for maximum tool offset of 999 by changing the D address to 3.0.

- Scroll down the **Word** column until you reach the address **D**.

Word	Leader/Code	Format	Trail Zero	Modal ?	Minimum	Maximum	Trailer
D	D 01	Numeric	<input checked="" type="radio"/>	<input checked="" type="radio"/> Yes <input type="radio"/> No	0	99	

2

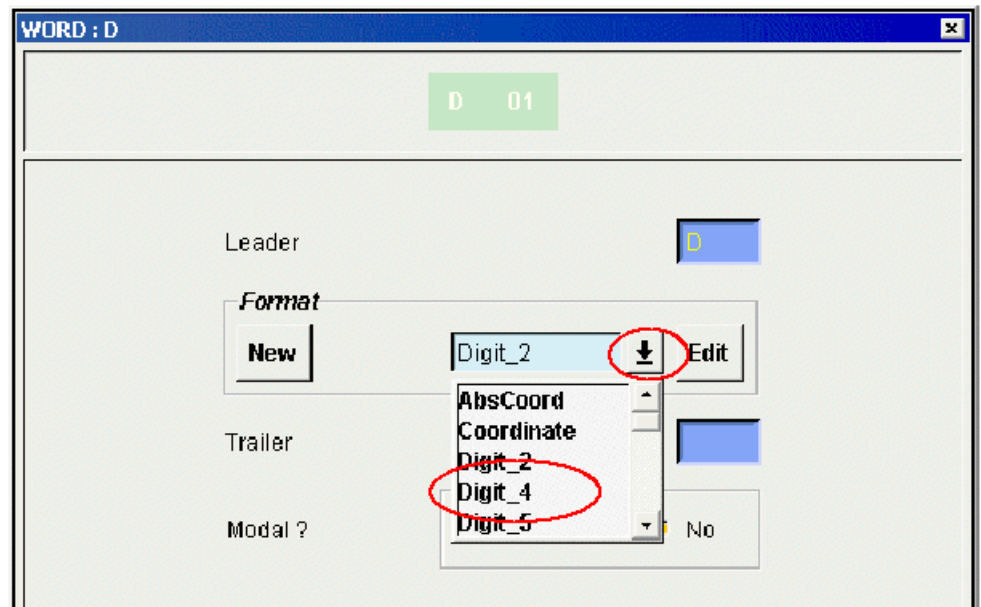
Notice that the range for the offset is from 0 to 99. The maximum number allowable since the format for the word is **Digit\_2**. By changing the format to **Digit\_4**, you can then allow for the maximum desired tool offset of 999.

- Click on the word, under the **Word** column heading.

The Word D: property page is displayed.

2

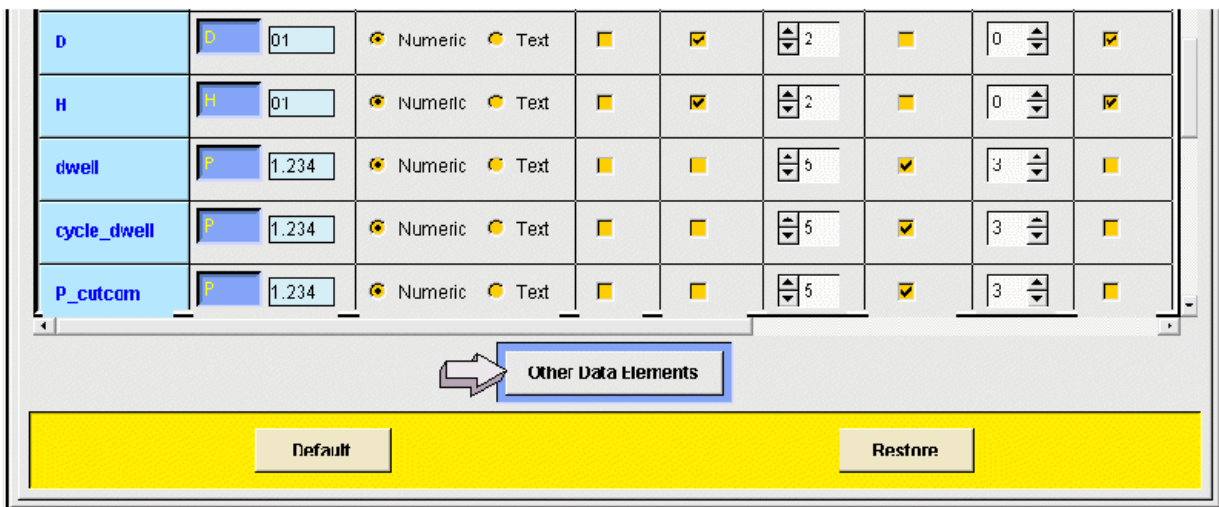
- Change the format from **2** to **4**  
select **Digit\_4** from the **Format** list.



- Change the number of available offsets from **99** to **999** by changing the **Maximum value** parameter from **99** to **999**.
- Choose **OK**.

**Step 6:** Change the sequence number to start at 5 and increment by 5 for each block afterwards.

- Select **Other Data Elements** from the **Word Summary** property page.

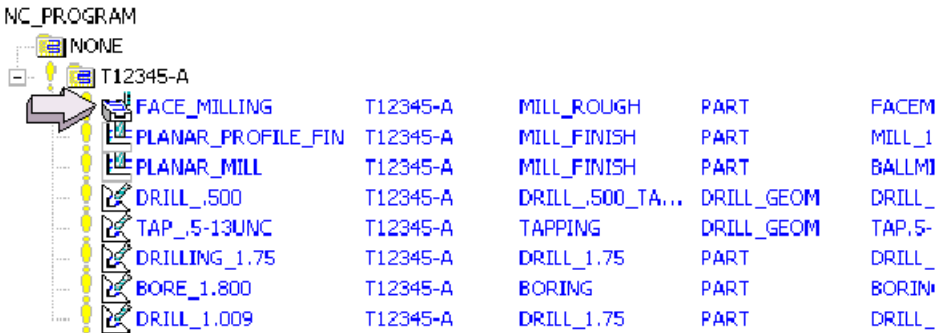


The Other Data Elements property page is displayed.

- ☐ Change the **Sequence Number Start** from **10** to **5**, then change the **Sequence Number Increment** from **10** to **5**.
- ☐ Choose **OK**.

**Step 7:** Save the post and run it against the test part, mill\_test.

- ☐ If necessary, enter the **Manufacturing Application** in NX.
- ☐ Using the Operation Navigation Tool, expand the **T12345-A** parent and select the **FACE\_MILLING** operation.



- ☐ Click **Post Process**.
- ☐ Click your post processor and select **OK**.

The default directory is write protected, you will need to change the output directory to your home directory.

- ☐ Change the default for Output File to your home directory and then click **OK**.
- ☐ Verify the output.

Your output will be similar to the following:

File	Edit
%	
N0005 G40 G17 G90 G70	
N0010 G91 G28 Z0.0	
N0015 T30 M06	
N0020 G00 G90 X19.5 Y-.2 S456 M03	
N0025 G43 Z4. H30	
N0030 Z2.2	
N0035 Z2.1	
N0040 G01 Z2. F13. D0030 M08	
N0045 X17.	
N0050 X-3.	
N0055 G00 X-5.25	

- ☐ Return to the **Post Builder**.

This completes the activity.

## Program & Tool Path Property page (continuation)

**Word Sequencing** allows the customization of the order of output of words comprising a block of NC data. The sequence order of the words will prevail throughout the post processor. If for example, the X and Y words were reversed, it will be reflected in all of the blocks generated by all Events. You can also suppress or make any word active. If **Balloon Help** is active, each word will display the context of that word. By default, a representative code is used to display that word, for example, the G17 word can also be G18 or G19. In other cases a D or T represents the adjust register or tool number register.

**Custom Command** allows the addition of user created custom commands. These are Tcl procedures that are executed by an Event. When creating a custom command, the **Post Builder** will place all of the correct syntax around the body of the procedure that you are creating. The first line of the Custom Command will be: **proc PB\_CMD\_name\_of\_proc {} {** and the last line will be **}**.

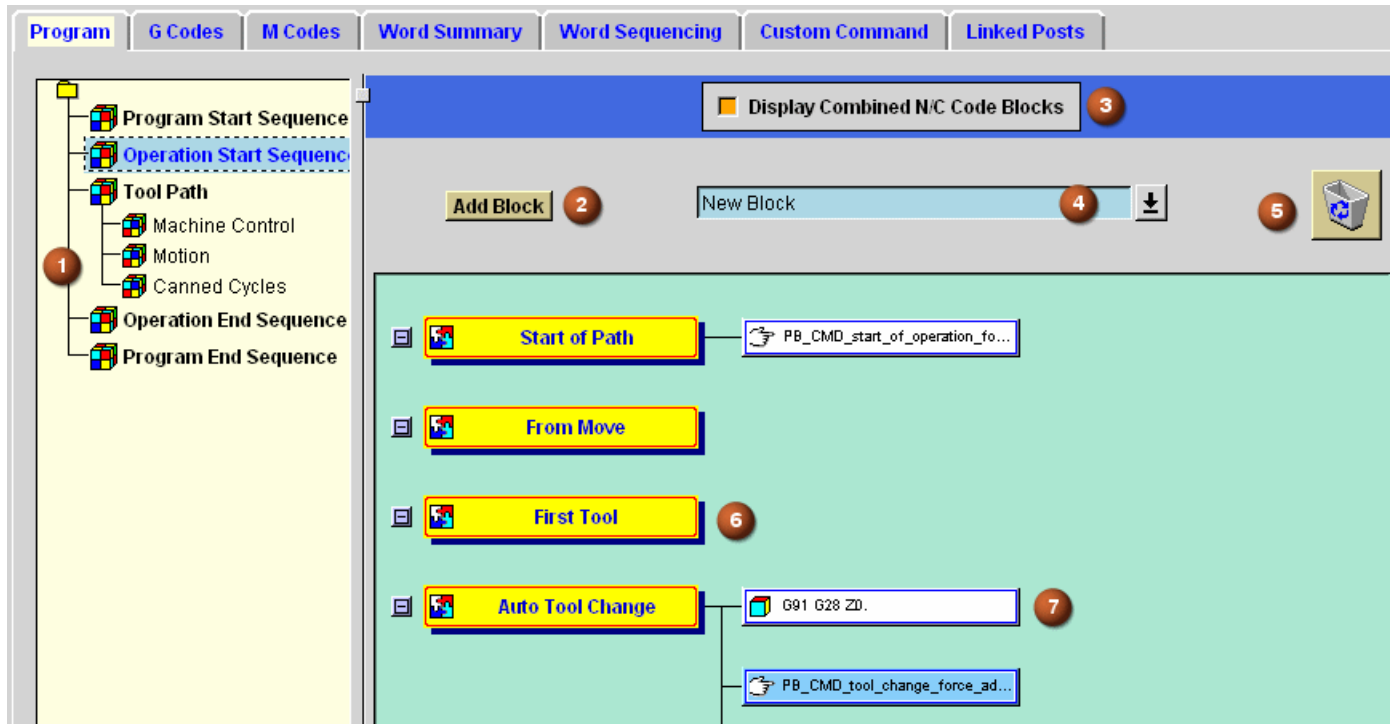
The **Custom Command** procedure will be processed in the sequence where it was placed. The **Post Builder** will check the validity of the syntax of Custom Commands. Numerous Custom Commands, designed for common "special applications", are supplied with every release of the **Post Builder**..

**Linked Posts** allows you to specify posts that can be linked to your current post. Linked Posts would be applicable in situations where separate posts have been developed for special machine options such as right angle milling heads, specialized mill-turn centers, machining centers having more than two rotary axis or any application that requires multiple post processors.



## About the interface

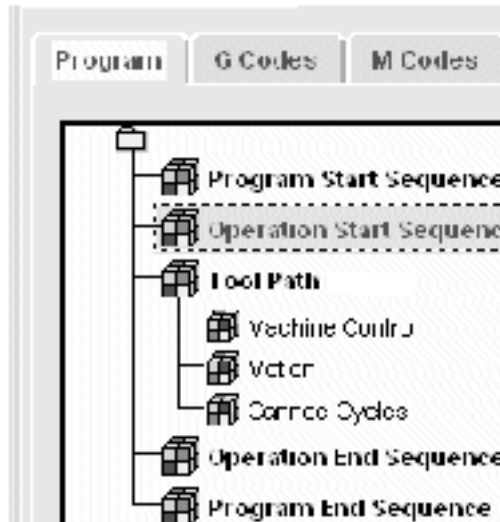
Under the **Program property page**, you will see two distinct windows displayed. The left most window, also referred to as the **Component Window**, contains a list of components that make up a program, displayed in a tree format. When a component is selected, the parameters associated with the component is displayed in the right most window, also referred to as the **Parameter Window**.



1. Program Sequence tree
2. Add Block
3. Display Combined Blocks
4. New Block
5. Trash Can
6. Sequence Marker
7. Sequence Block

A detailed explanation of the above numbered components follows:






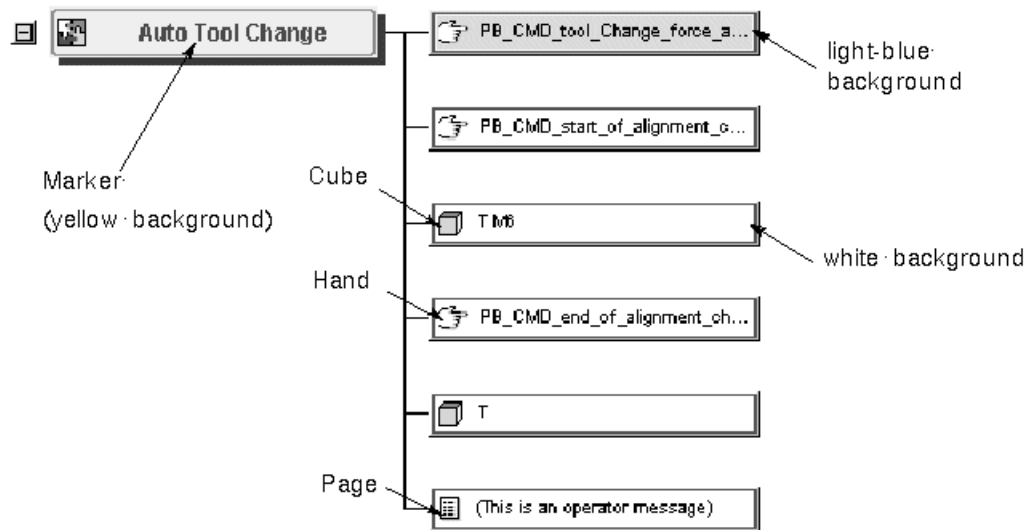


The Program contains the following sequences;

An NC program has a beginning and an end with a number of operations or steps in between. Collectively, the program is considered to be composed of a sequential order of Events. Due to the way that tool paths are created within NX, certain Events, such as Tool Change, Spindle Start or Approach Motion Events can occur at the start and end of an NC program and for each operation. At the **markers** (designated as yellow rectangular blocks) of these pre-determined Events, you can force certain instruction blocks to be output. This is referred to as a **sequence**. Understanding the sequences and the events that trigger them allow you to properly place custom commands and blocks to obtain the desired output.

Three types of components can be attached to the **markers** of the various sequences. They are identified by the following icons:

- **Cube:** a normal block 
- **Hand:** a custom or MOM command 
- **Page:** an operator message 



When a rectangular block appears light-blue in color, it designates the component as being used by more than one marker. Changing one instance will affect all instances of that component.

When a rectangular block appears white in color, it designates the component as being used only once.

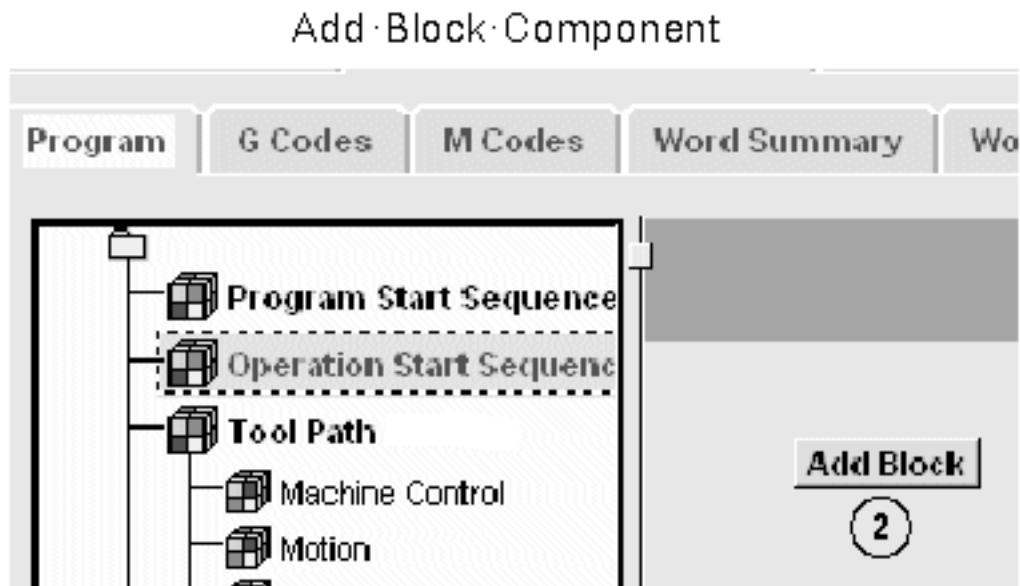
From the post processor's view, an NC program consists of five sequences and a collection of Events that can occur in each operation. They are as follows:

- **Program Start Sequence** which determines which blocks are output when this particular Event occurs before the tool path is read. This Event occurs before any other Event is processed.
- **Operation Start Sequence** contains Events and actions that will occur from the start of an operation to the first cut motion. Every operation will have a first tool change, automatic tool change, manual tool change or no tool change.
- **Tool Path Sequence** consist of Events and actions that pertain to machine control, motion control and canned cycles.
- **Machine Control** options control items such as coolant, spindle, tailstocks and clamps. They can also be used to change modes such as incremental or absolute, inverse time, feed per minute/per revolution and constant surface speed. Information can be passed to a post processor by parameter data which is available at the Start of Operation. This includes feeds and speeds and any cutting parameters such as stepover and tolerances. Information can also be passed by way of User Defined Events-UDEs- (sometimes referred to as **post commands**). UDEs specified as Start of Post come out after the Start of Operation in the order specified by

Operation Markers (a marker indicates Events that may occur). UDEs associated with the tool, program, method, or any other parent group are output before the Start of Operation Event. The recommended method of output for the Start of Operation NC code is to organize all of the data (spindle speed, spindle direction, coolant status, length compensation, tool number) and then output this data as part of the Operation Start Sequence.

- **Motion** options describe how the post processor will process GOTO records in the tool path. All motion generated with a feed rate of zero is processed by the Rapid Move Event. If any motion types have a non-zero feed rate, the Linear Move Event will be used. Motion types included in this Event type are cut, engage, first cut, stepover and side cut. Circular motion is handled by the Circle Move Event. An Event type for Nurbs is currently handled by the **Post Builder** for Fanuc 6M, Siemens and Heidenhain controls.
- **Canned Cycle** options describe how the post processor will handle canned cycles. You can define and or modify the G-code, associated parameters, and output blocks that will be used with pre-defined cycles.
- **Operation End Sequence** contains Events and actions that occur from the final retract motion to the end of the operation. These are items like returning to a home position or turning off the spindle and or coolant. If the same End of Operation function is always going to be performed at the end of every operation you can enter it into this sequence instead of programming UDEs.
- **Program End Sequence** contains Events and actions that occur from the end of the last operation to the end of the program.

By activating the **Add Block** button, you can drag and drop a **block component** into the location desired. The block component can be placed above, below or next to an existing block component. Block components which are available are listed in the **Add Block component available** option menu.



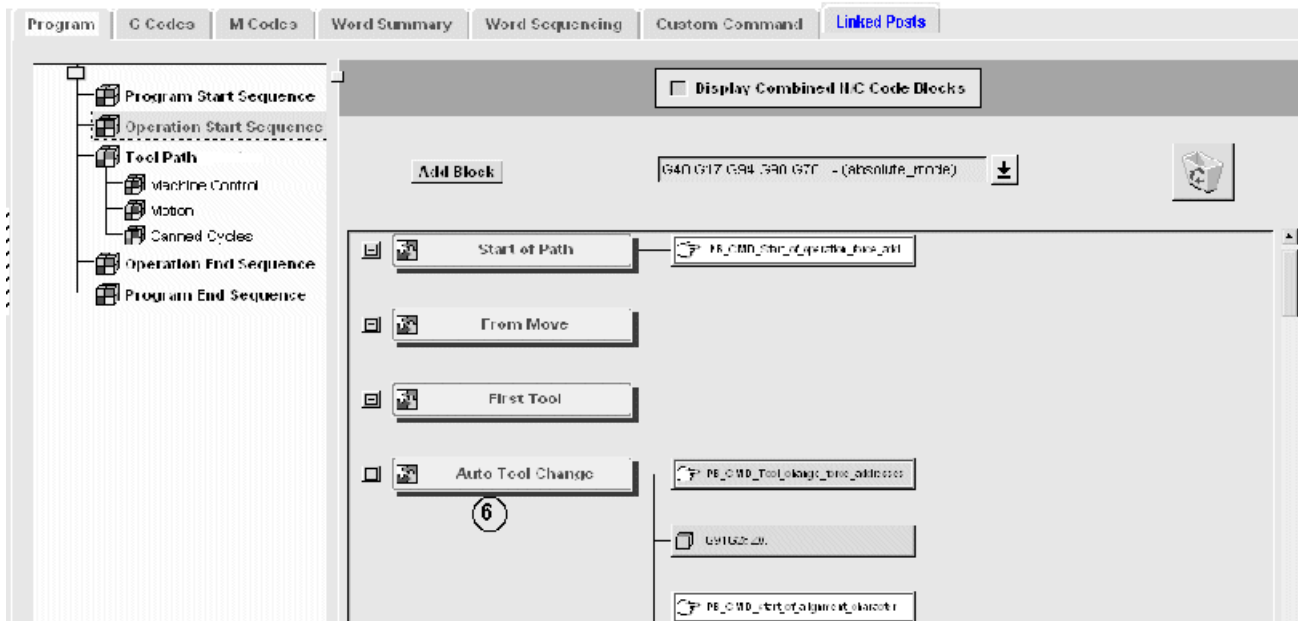
Activating the **Display Combined N/C Code Blocks** toggle button will display all block components in the form of NC codes. By default, block components are displayed in terms of their descriptions.

The **Add Block** component option menu describes the blocks which are available. You can create a new block component and apply it to a sequence. You may also select one of the pre-defined block components.

The **Trash** bin is used to delete any unwanted block component by dragging the block component to the trash bin icon.

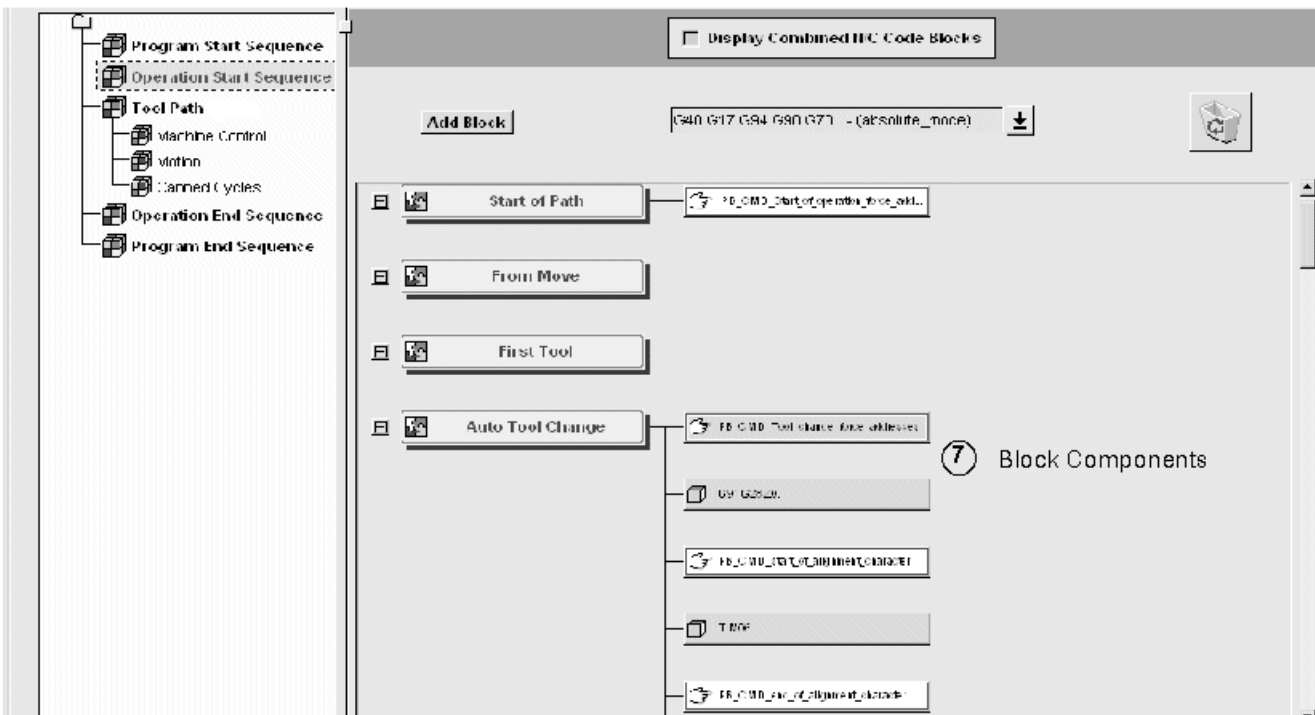
You can also delete the component by right clicking on the component and selecting delete from the drop down menu.

The **Operation Start** sequence is one of the sequences which can occur with each operation (one of the five sequences, **Program Start**, **Operation Start**, **Tool Path**, **Operation End** and **Program End**). Yellow colored blocks always represent the sequence blocks (also referred to as markers).



You can neither add or delete Event blocks.

A block component can be placed above, below or next to an existing block.



If you right click on the block component, a pop-up menu appears which allows you to **Cut**, **Copy As** or **Delete** a custom command block. If a block contains NC data, your options are **Edit**, **Force Output**, **Cut**, **Copy As** or **Delete**.

## Activity — Program & Tool Path parameters

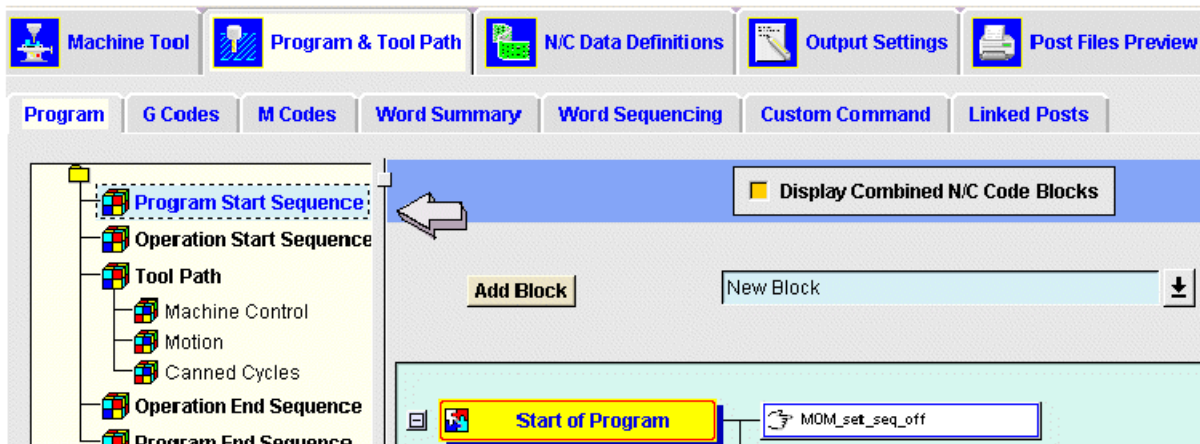
In this activity you will examine and modify Program & Tool Path Parameters.

**Step 1:** Enter the Program and Tool Path section of **Post Builder**.

- ☐ Click **Program & Tool Path**.
- ☐ If necessary click **Program**.

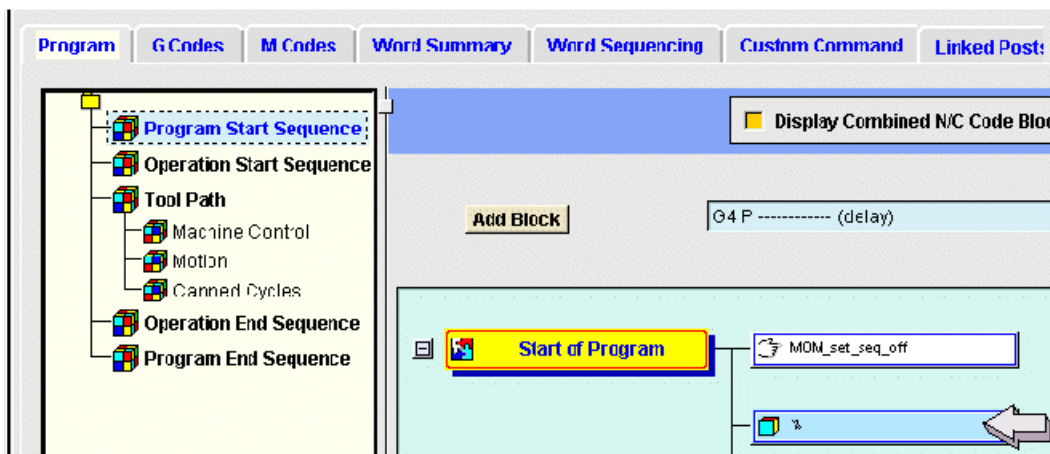
**Step 2:** Create a Program Start Sequence with a Rewind Stop Code of (#).

- ☐ Click **Program Start Sequence** from the Component Window.

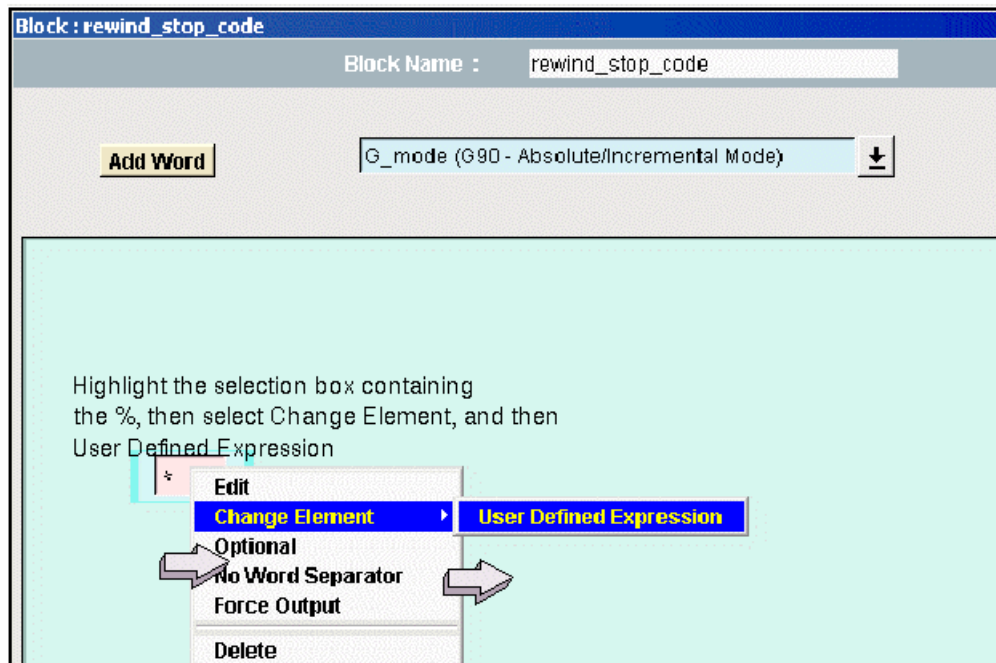


- ☐ Double-click the **block component** containing the % from the Sequence Window.

Prior to selecting this particular component, notice the light-blue color of the block component. This indicates that this particular block is used again in either the same or another sequence. In this particular example, this block component appears again in the **Program End** sequence.



The **Block: rewind\_stop\_code** property page is displayed.



- ☐ Highlight the selection box containing the "%", right-click, select **Change Element**, then select **User Defined Expression**.

The Expression Entry text box is displayed.

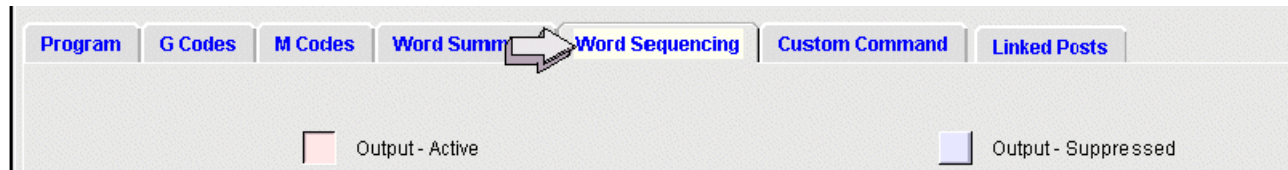


- ☐ Change the "%" to "#".
- ☐ Click **OK** from the **Expression Entry** text box.
- ☐ Choose **OK** from the **Block: rewind\_stop\_code** property page.

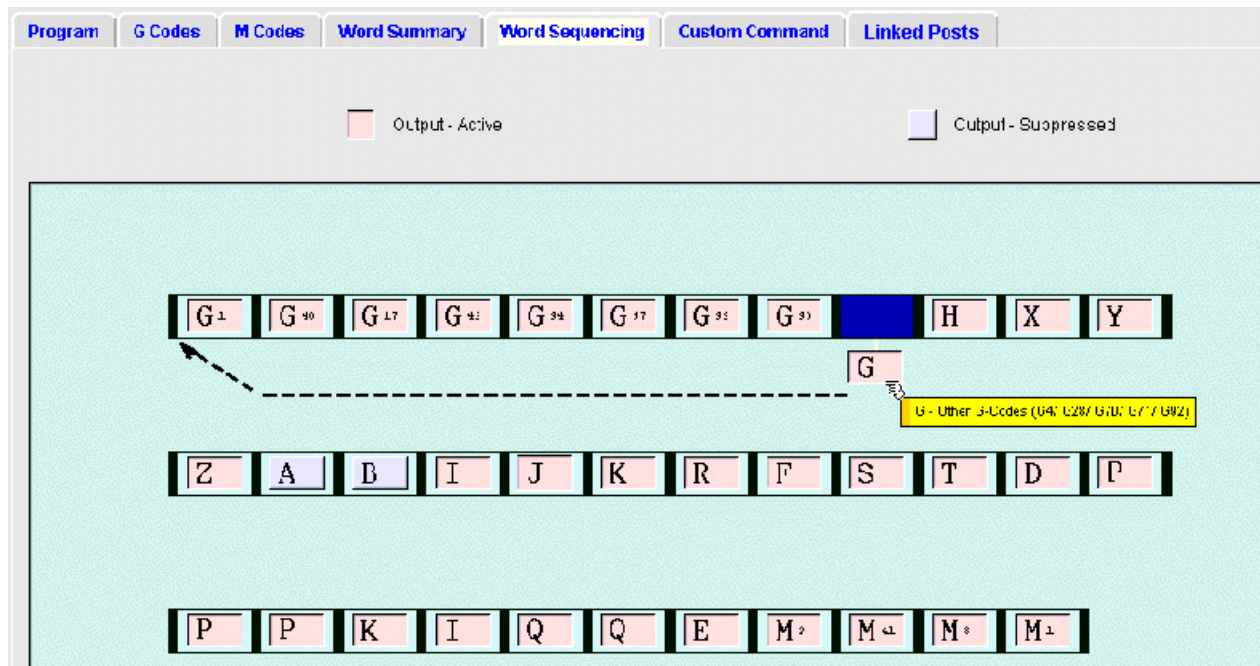
As mentioned earlier in this activity, this block component also appears in the **Program End** sequence. A change to one block component will change all like components, regardless of their location.

**Step 3:** Create a Program Start sequence with a G70G90G40G17G94 block by modifying the Word sequence (NOTE: list the G codes in the order shown).

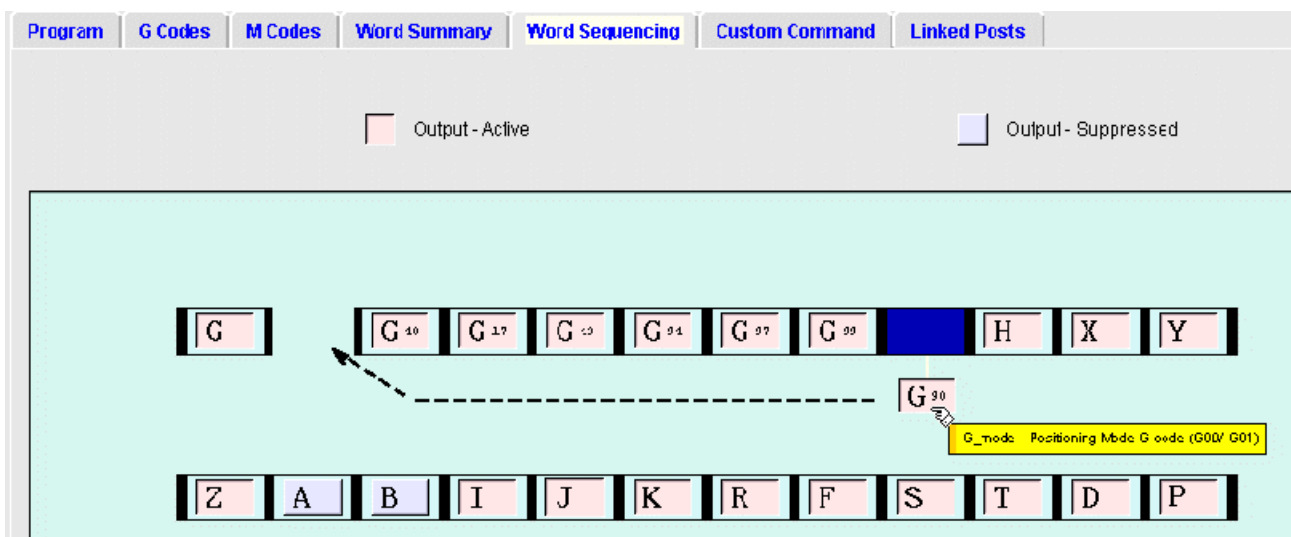
- ☐ Click **Word Sequencing**..



- ☐ Click the "**G**" block and "drag" it to the beginning of the line.

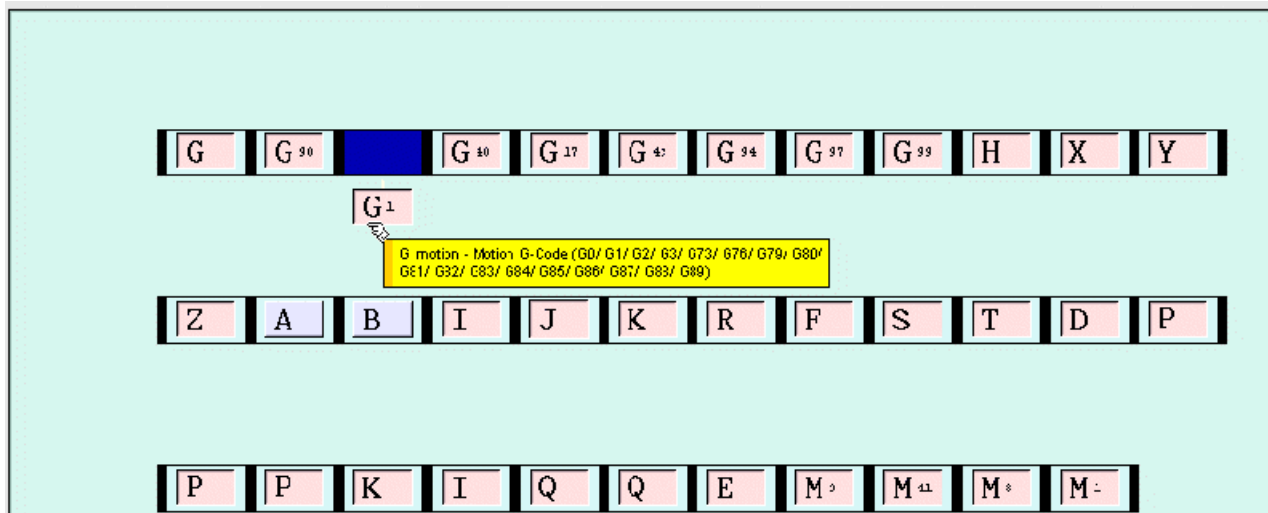


- ☐ Click the "**G90**" block and "drag" it between the "G" and "G40" block.





- ☐ Select the "**G01**" block and position it between the "**G17**" and "**G43**" block.



**Step 4:** Suppress the output of selected words.

- ☐ Click "**G97**" once to suppress the "**G97**" output (turns from pink to blue).
- ☐ Repeat the previous action and suppress the following: "**G99**", the three "**P**" words and the "**E**" word.

**Step 5:** Change the order of the "D" and "H" words.

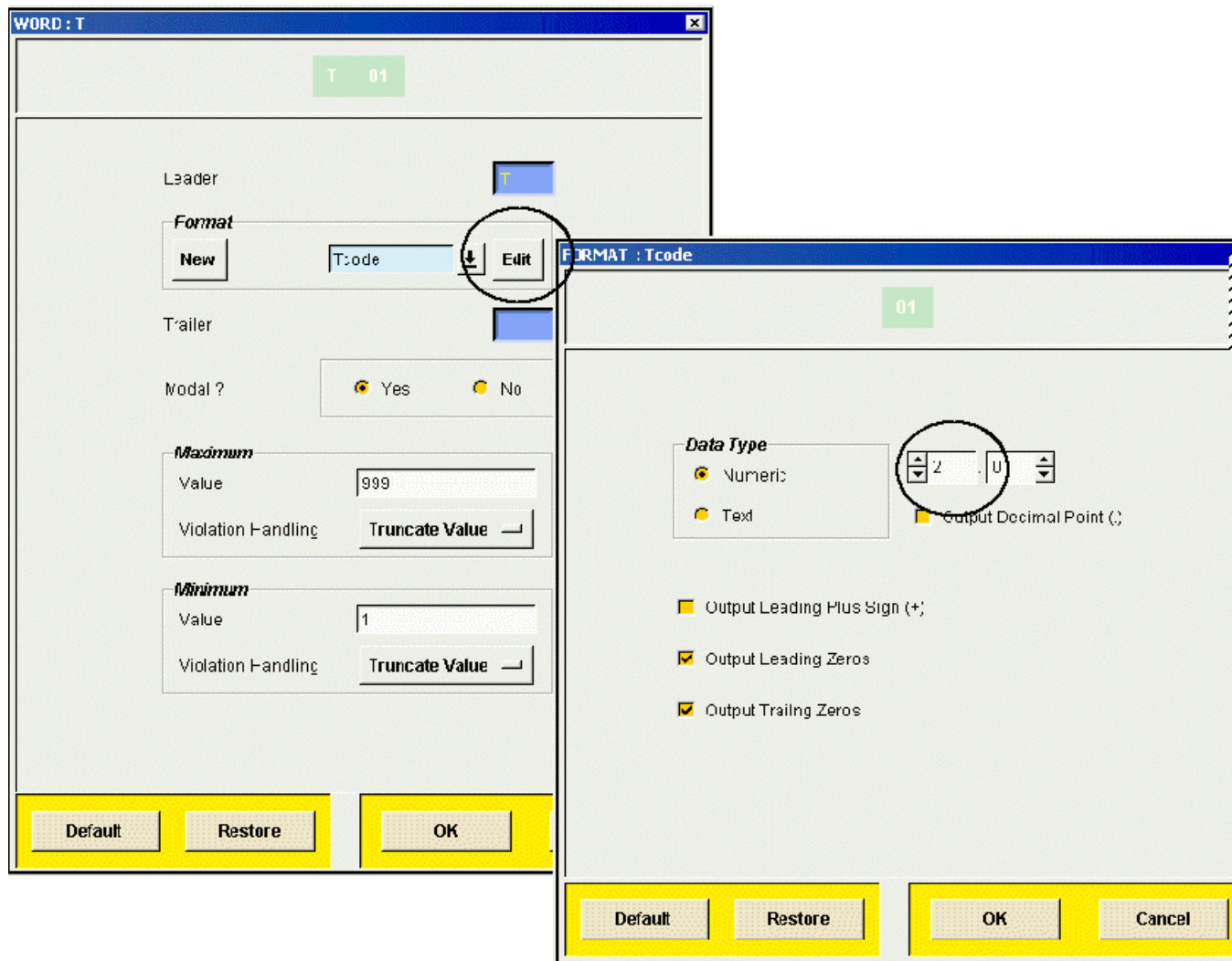
- ☐ Move the "D" word behind the "H" word.

**Step 6:** Change the tool parameter number to a maximum of 999 and minimum of 1.

- ☐ Right-click "T" and select **Edit**.
- ☐ Change the maximum value from **99** to **999**.
- ☐ Change the minimum value from **0** to **1**.

2

- ☐ Change the Tcode output format by selecting Edit under Format and change the output field from **2** to **3**.




- ☐ Choose, **OK** twice.
- ☐ **Save** the post processor.


**Step 7:** Save the post and run it against the test part, mill\_test.


- ☐ If necessary, enter the **Manufacturing** application.









- Using the Operation Navigation Tool, expand the **T12345-A** parent and click **FACE\_MILLING**.

NC\_PROGRAM

 NONE

 T12345-A

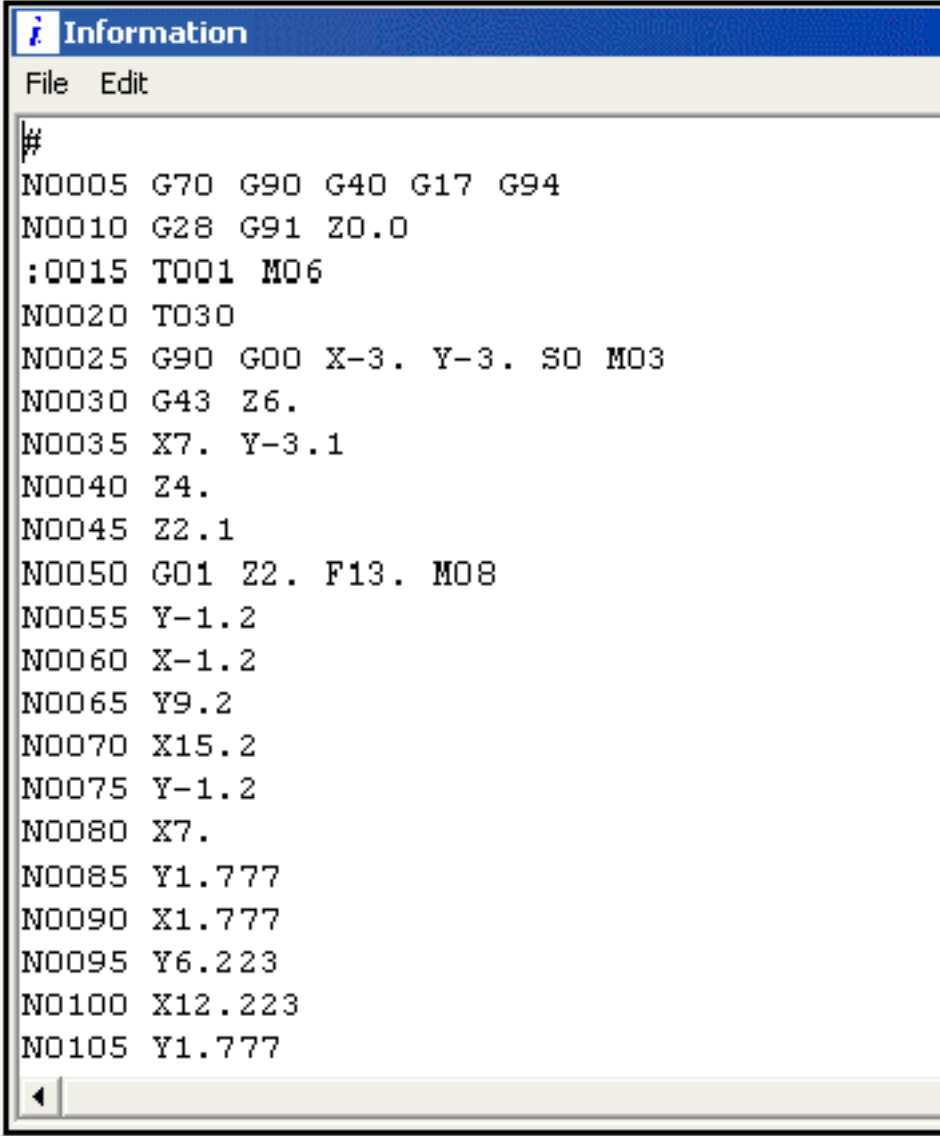


	FACE_MILLING	T12345-A	MILL_ROUGH	PART	FACEMILL_4.00	Yes	Milling Toc
	PLANAR_PROFILE_FIN	T12345-A	MILL_FINISH	PART	MILL_1.375	Yes	Milling Toc
	PLANAR_MILL	T12345-A	MILL_FINISH	PART	BALLMILL_.875	Yes	Milling Toc
	DRILL_.500	T12345-A	DRILL_.500_TA...	DRILL_GEOM	DRILL_.500	Yes	Drilling To
	TAP_.5-13UNC	T12345-A	TAPPING	DRILL_GEOM	TAP.5-13UNC	Yes	Drilling To
	DRILLING_1.75	T12345-A	DRILL_1.75	PART	DRILL_1.75	Yes	Drilling To
	BORE_1.800	T12345-A	BORING	PART	BORING_BAR...	Yes	Drilling To
	DRILL_1.009	T12345-A	DRILL_1.75	PART	DRILL_1.009	Yes	Drilling To



- Click **Post Process**.
- Select your post processor and select **OK**.
- Accept the default for **Output File** and then select **OK**.
- Verify the output.

Your output should be similar to the following:



```
#
N0005 G70 G90 G40 G17 G94
N0010 G28 G91 Z0.0
:0015 T001 M06
N0020 T030
N0025 G90 G00 X-3. Y-3. S0 M03
N0030 G43 Z6.
N0035 X7. Y-3.1
N0040 Z4.
N0045 Z2.1
N0050 G01 Z2. F13. M08
N0055 Y-1.2
N0060 X-1.2
N0065 Y9.2
N0070 X15.2
N0075 Y-1.2
N0080 X7.
N0085 Y1.777
N0090 X1.777
N0095 Y6.223
N0100 X12.223
N0105 Y1.777
```

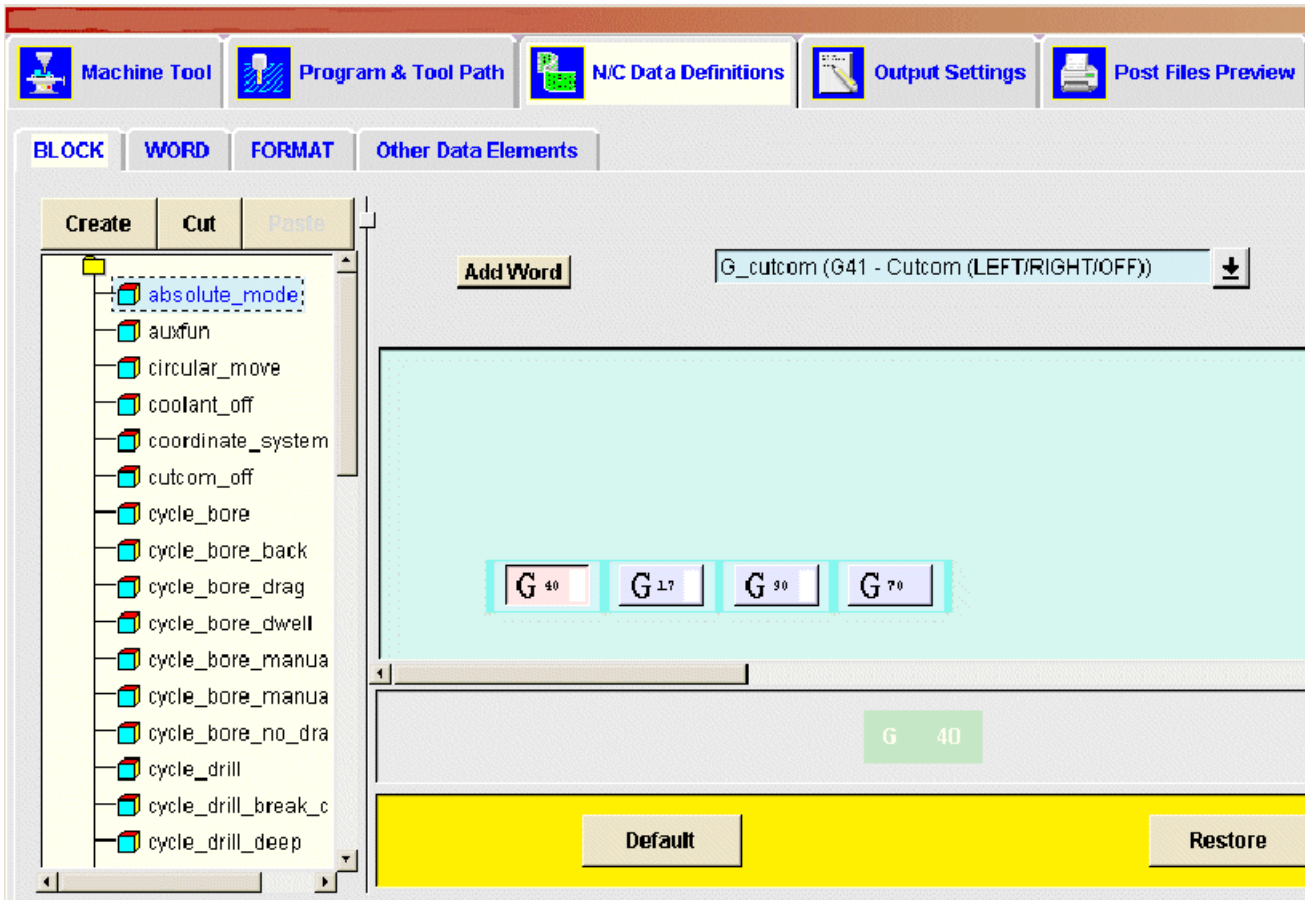
Return to the **Post Builder**.

This completes this activity.

## NC Data Definitions property page

The **NC Data Definitions** property page allows the definition of output formats for NC data.

2



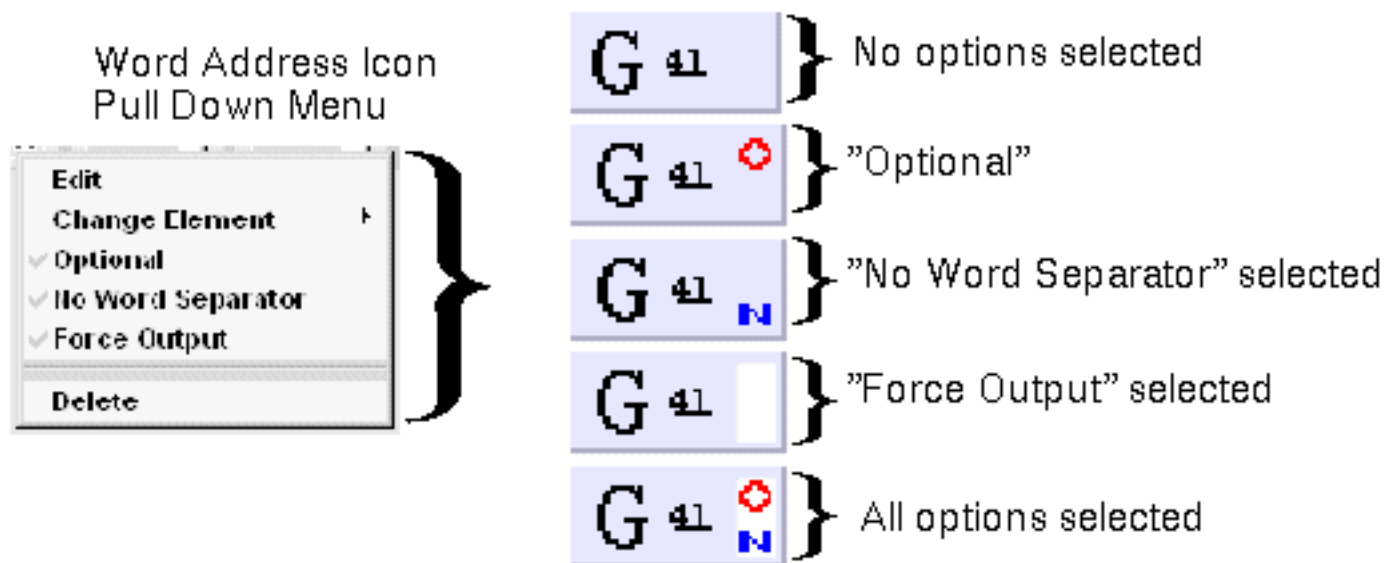
Data definitions are broken down into four sub classifications, also represented in tabular format. These sub classes are:

- **Block** defines the format sequence of the instruction data that appears as output. Blocks are made up of word elements, each being defined by an address and variable. There are two methods used to create a block. From the sequence or event property page you may drag a blank block and drop it into a sequence or event. You can also edit any existing block with the sequence, event or NC Data Definitions block property page. You may then edit or build your block of data.
- **Word** defines the address, output format and the structure of the words that comprise the instructions. Parameters associated with address are format, maximum and minimum values, modality and leading and trailing characters. A word is composed of an address leader, a number or text and a trailer. An address leader may be any character. The address leader is usually a single character such as G, M, X, Y, Z, etc. The trailer is usually a blank character. You can specify the format for current words

from an existing list of formats or edit one of the existing formats. You can also create a new format from the Format section of this property page.

- **Format** for data output can be defined as **real**, **integer** or **character string**. Parameters for **Format** vary according to the data type selected. **Real** numbers are generally used for co-ordinate values; **integers** are used for registers and **character strings** are used for commentary messages and special output.
- **Other Data Elements** allow for the control of sequence number output, the use of special characters for word separation, end of block and message start and end codes.

You have probably noticed that block word address icons have symbols associated with them. These various icon symbols are **switched** on when you pass the mouse over the word address icon, right-click and select the pull down menu. Choosing **Optional**, **No Word Separator**, **Force Output** or any combination of the three from the pull down menu will turn the various symbols on.



A checkmark before the option is an indicator of the option being turned on.

- **Force Output** forces the selected code for output within the current block only. This is useful, for example, to force out a T or some other code for each tool change. By default all words are modal and will output only if the value changes from one block to the next. **Force Output** can only be specified within the context of an Event or sequence.
- **No Word Separator** results in the separator character being suppressed following the word on output. This is used mainly for CLS output.

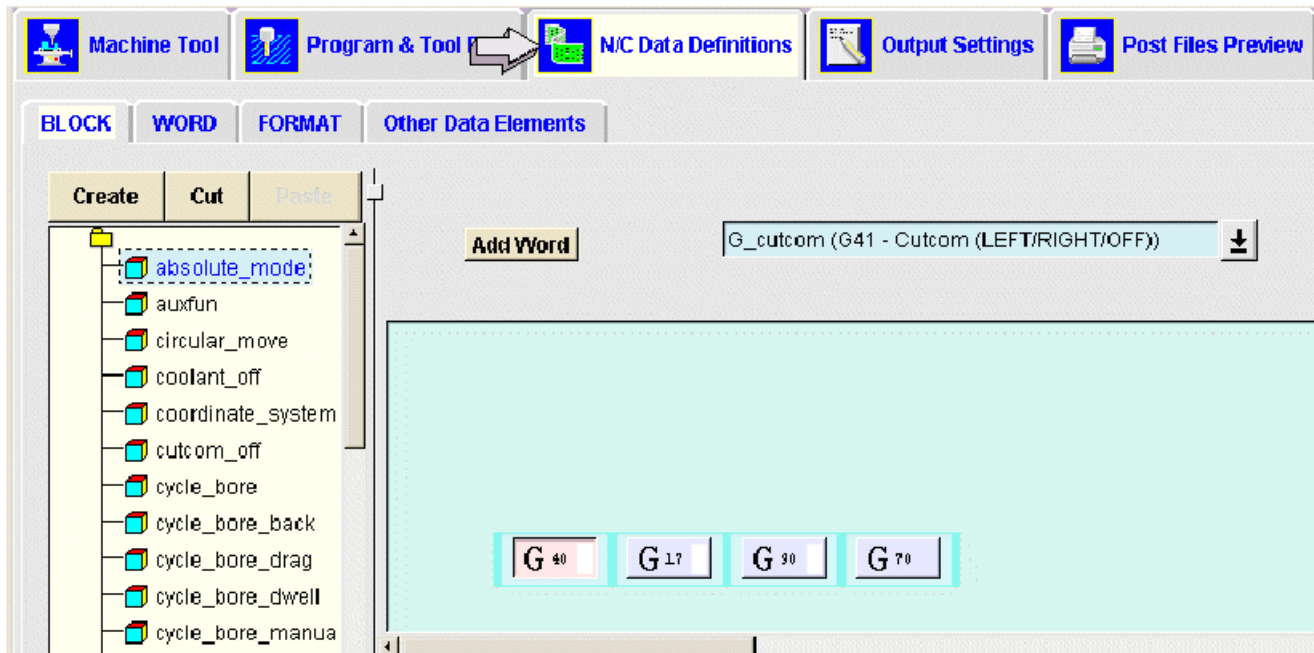
- **Optional** will test to see if the address has a variable defined, if defined, it will be used; if the variable is not defined, the address will be suppressed. Note that this option takes priority over the "Force Output" option.

## Activity — NC Data Definition

In this activity you will modify a word address to force a “G91” code and then modify the existing “G84” tapping code to produce a “G84.1” rigid tapping cycle.

**Step 1:** Enter NC Data Definition section of the **Post Builder**.

- ☐ Click **NC Data Definition** from the **Post Builder** selection menu.



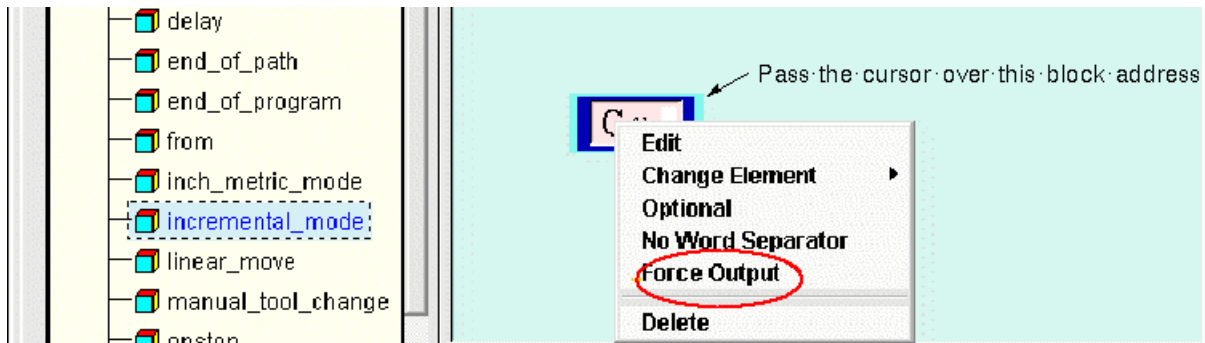
- ☐ If necessary, click **Block**.



**Step 2:** Change the incremental mode block address variable to force out a G91 code.



- ☐ Click the **incremental\_mode** variable and then right-click G91 and select **Force Output**.

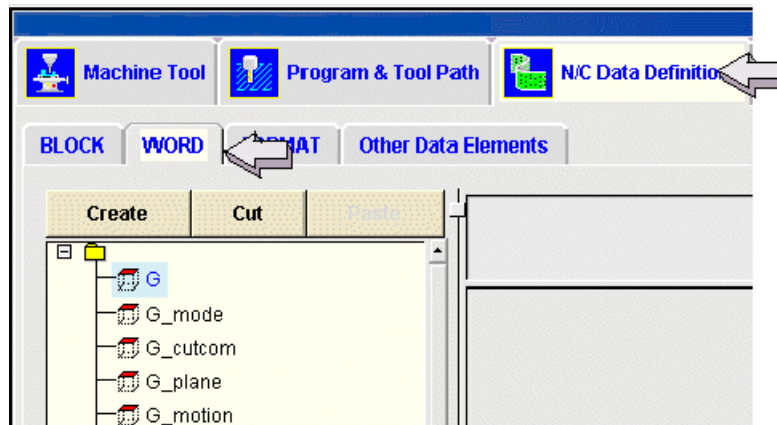


- ☐ Notice the white bar, indicating **Force Output**, that appears on the block address icon.

**Step 3:** Create a G84.1 Rigid tapping sequence.

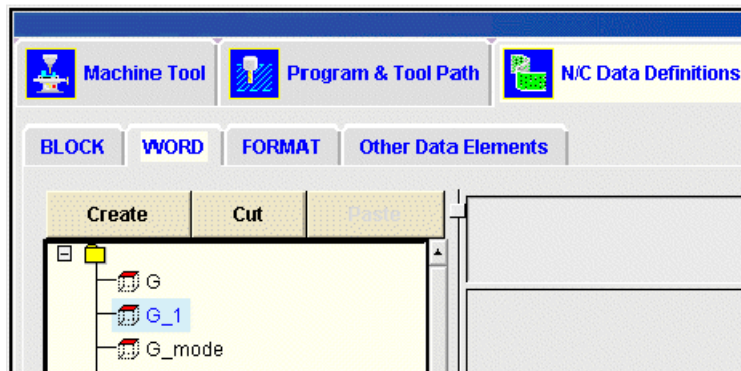
You will now modify the existing **G** code that will be output for a tapping cycle. You will change the standard **G84** code to **G84.1** used in a rigid tapping sequence.

- ☐ If necessary click **N/C Data Definition**.
- ☐ Click **Word**.



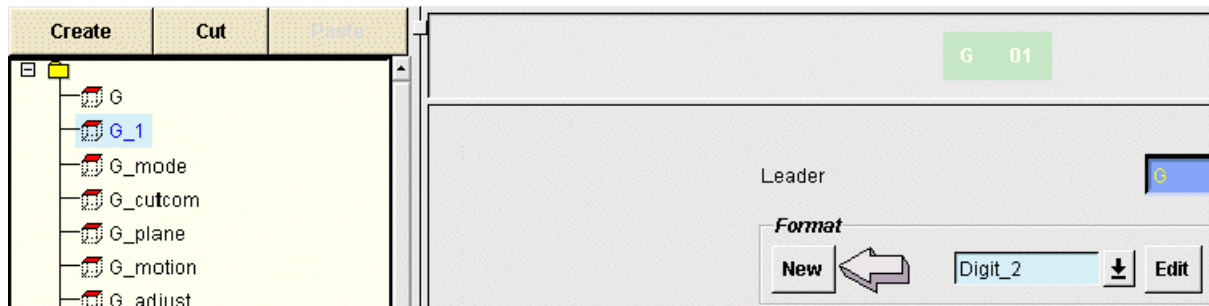
- ☐ Select the **G** block, right-click, select **Create**.

The **G\_1** block is created.

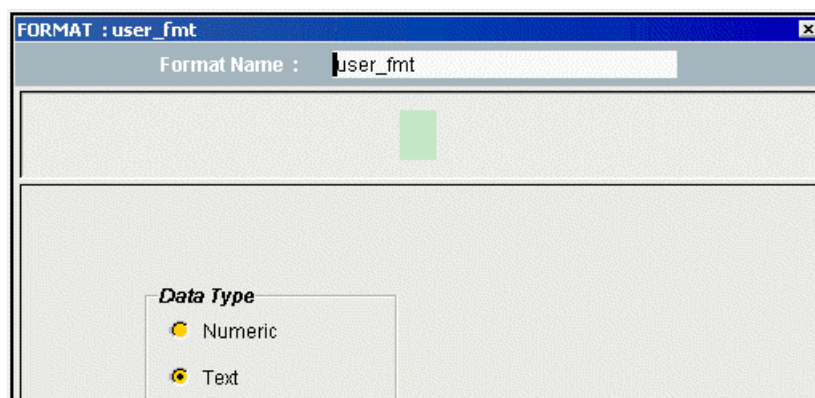


The standard tap code is **G84** which represents a two digit integer. You will be creating a new tap code, **G84.1**, which represents a three digit, real number. You will now create a new format to represent the new tapping code of **G84.1**.

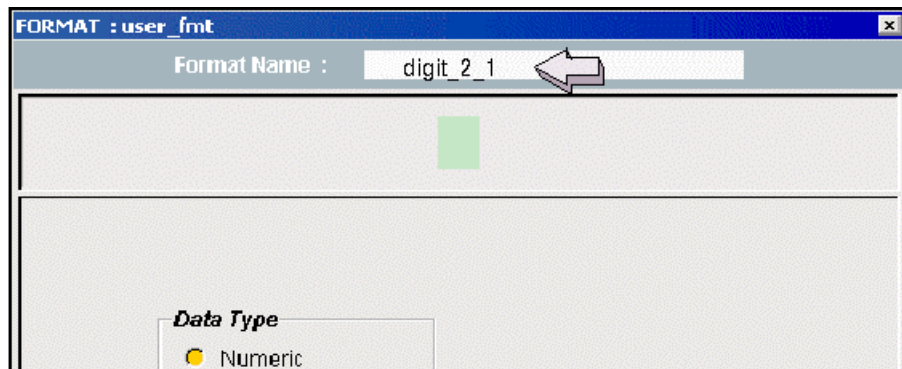
- ☐ Click **New** under Format.



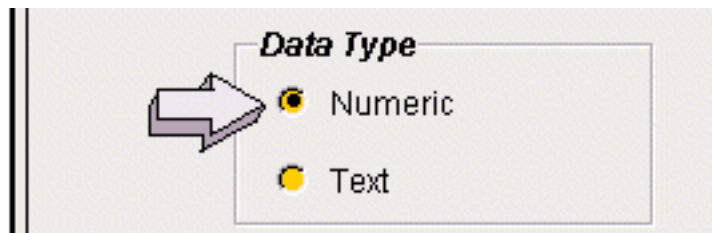
The **Format: user\_fmt** property page is displayed.



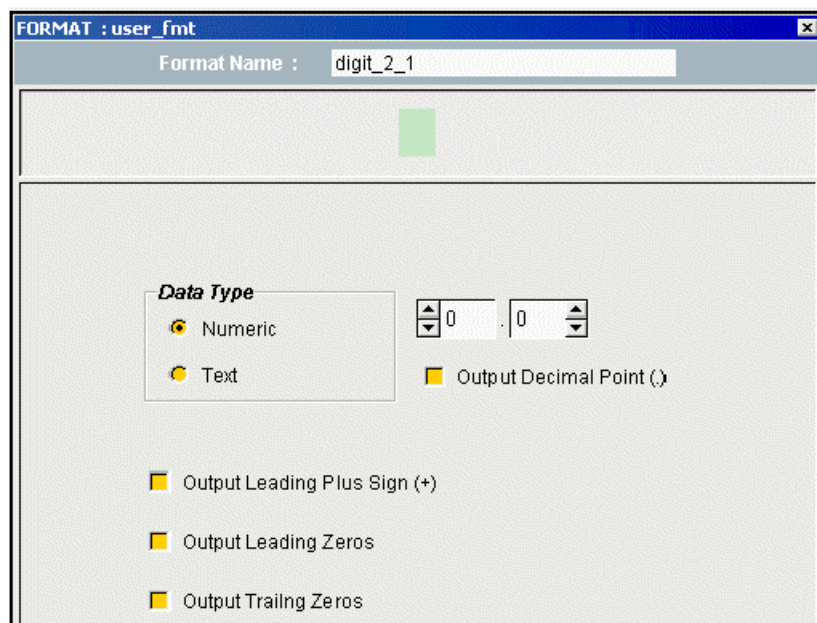
- ☐ Type **digit\_2\_1** in the **Format Name** text box.



- ☐ Select **Numeric** under **Data Type**.



The **Format: user\_fmt** property page displayed is expanded.



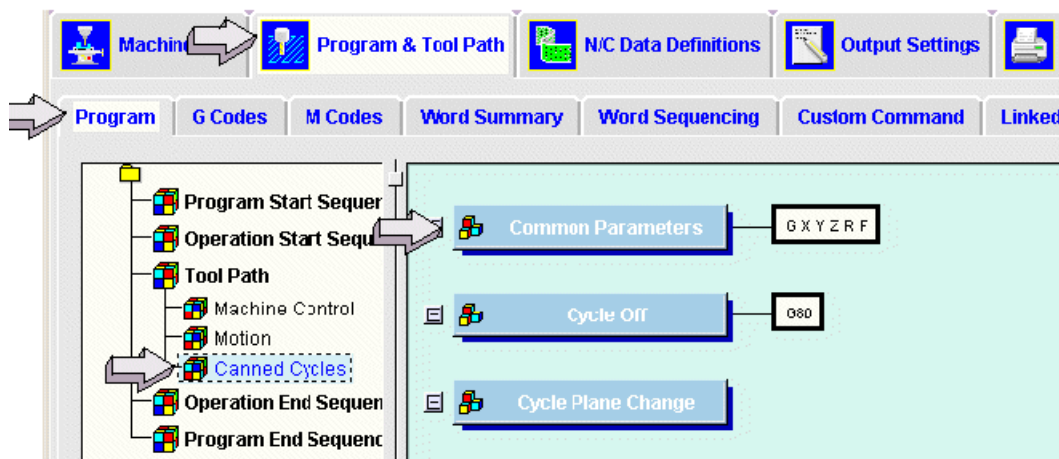
- ☐ Change the output to two places to the left of the decimal and one place to the right.

- ☐ Select the **Output Decimal Point** box.

- ☐ Click **OK**.

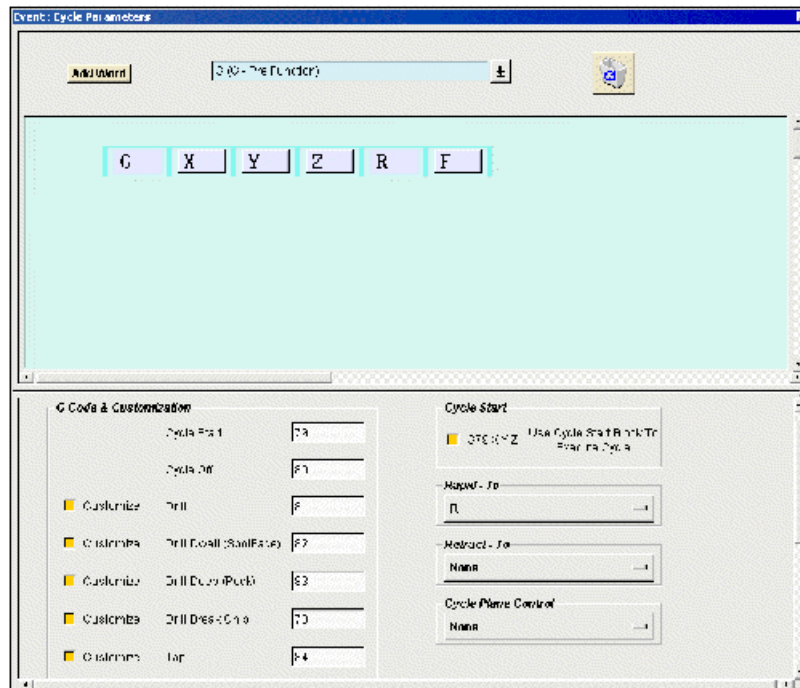
You have just completed setting the format for the **G84.1** code. You must now set the parameters to allow customization of the tapping code.

- ☐ Click **Program & Tool Path**.
- ☐ If necessary, click **Program**.
- ☐ Click **Canned Cycles** and then click on **Common Parameters**.



The **Event: Cycle Parameters** property page is displayed.





- ☐ Under the **G Code** and **Customization** section, scroll down to **Customize Tap** and check the box.

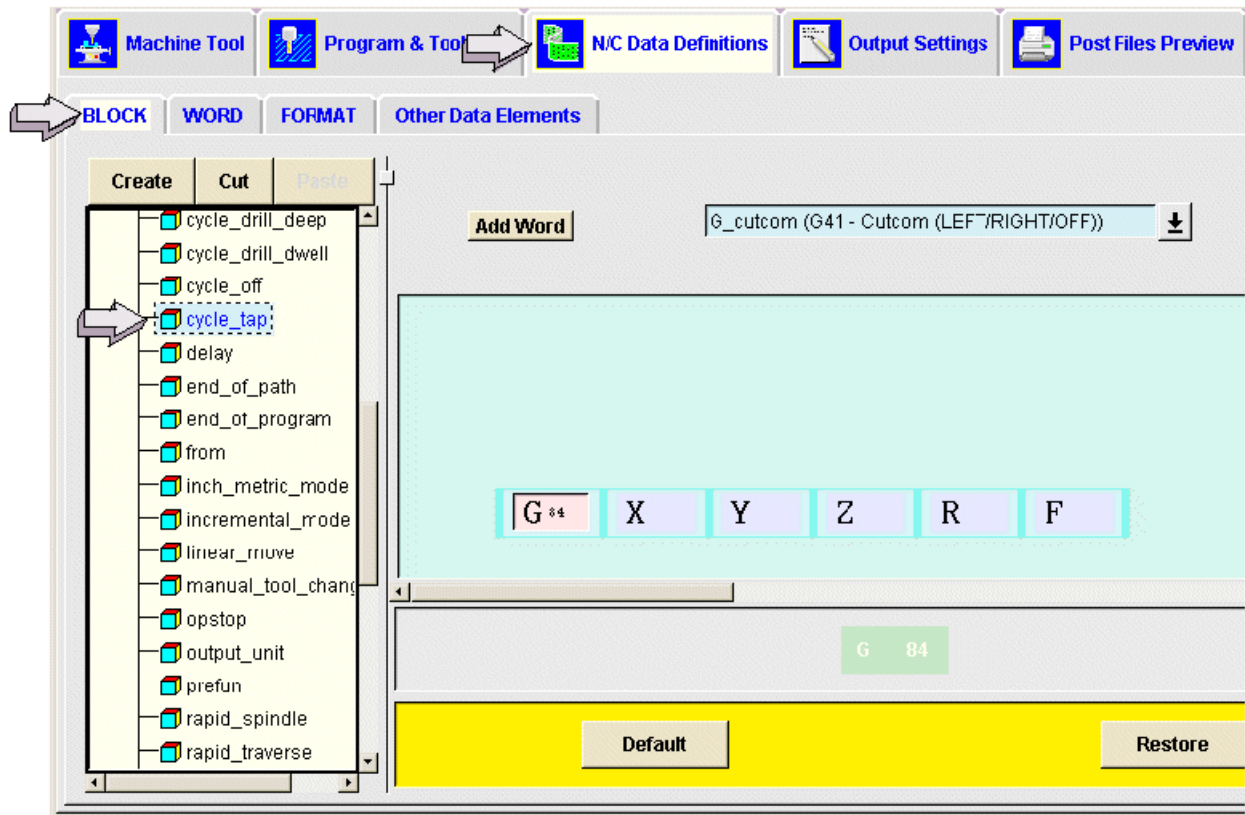


- ☐ Click **OK**.

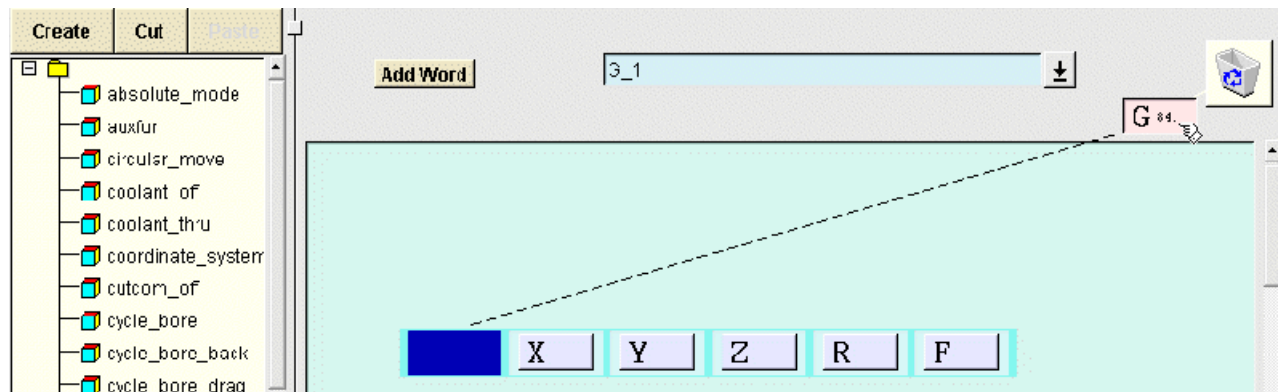
You must now discard the previous tapping word block and create a tapping word block that allows for the **G84.1** word.

- ☐ Click **N/C Data Definitions**.
- ☐ Click **Block**.
- ☐ Select **cycle\_tap**.

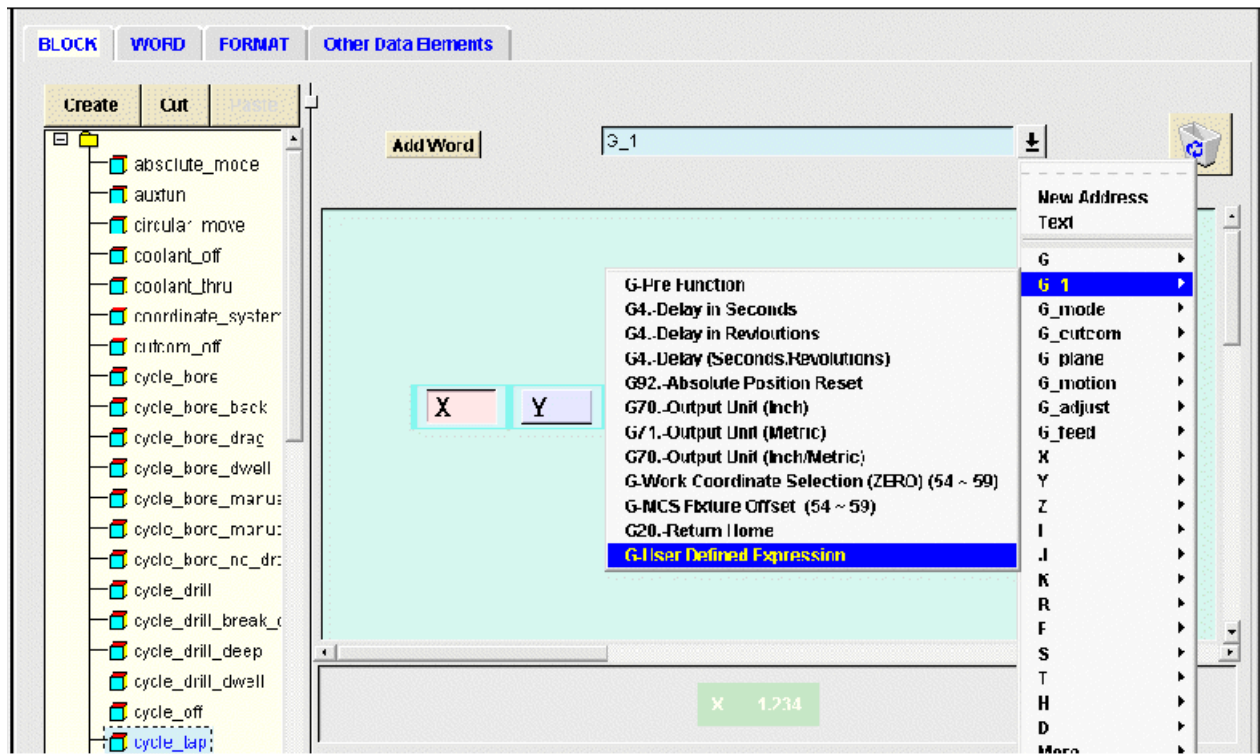
2



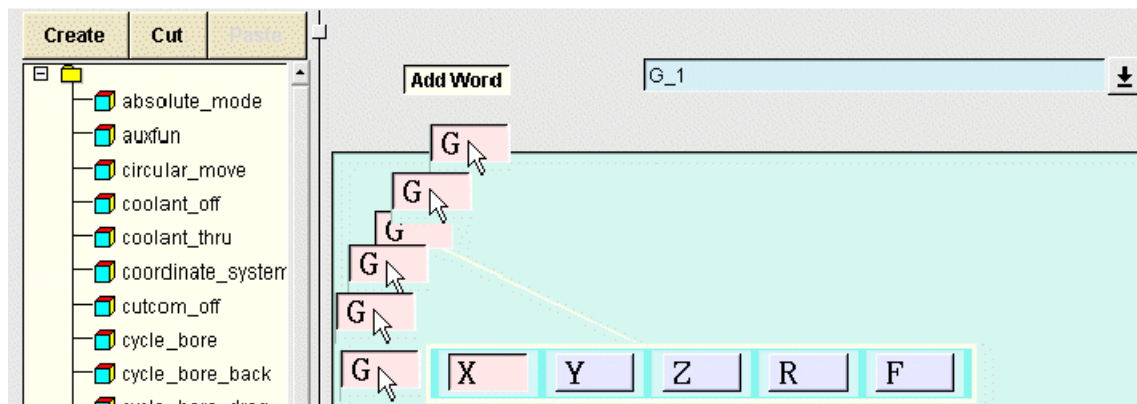
- ☐ Drag the **G84** block to the trash bin.



- ☐ Click **G\_1** from the **Add Word** list , then choose **G-User Defined Expression**.



- ☐ Click **Add Word** and drag the block to the beginning of the line.



The **Expression Entry** dialog box is displayed.

- ☐ Type **84.1** into the **Expression Entry** box.
- ☐ Click **OK**.

Save the post and run it against the test part, mill\_test.

- ☐ If necessary, enter the **Manufacturing application** in NX.
- ☐ Expand **T12345-A** and select the **TAP\_.5\_13UNC** operation.



- ☐ Click **Post Process** .
- ☐ Select your post processor and select **OK**.
- ☐ Accept the default for **Output File** and then click **OK**.
- ☐ Verify the output.

Your output should be similar to the following:

G84.1 X1. Y1. Z1.1 R2.3 F67.9

- ☐ Return to the **Post Builder**.

This completes this activity.



## Create new M or G code groups

M and G codes are normally classified into logical groups. One such group is the **Motion** group. This group consists of the words G00 for rapid, G01 for linear feed, G02 for circular interpolation CW, G03 for circular interpolation CCW and G80 thru G89 codes that represent drill cycles. Only one word from this group can be active at any given time, i.e. it is illogical for a machine to be moving in the G00 mode (rapid) while making a G01 move (linear feed) simultaneously.

As a general rule, **Post Builder** supports all common M and G code groups. In some instances however, certain machine tool/controller combinations support functions that are unique and require custom M or G codes. **Post Builder** allows you to create these new M or G code groups as will be shown in the following activity.

## Activity — Create a new M-Code group

In this activity, you will create a new M-code group to support a machine tool feature of coolant through the tool. In a later activity, you will modify the coolant UDE (User Defined Event) to support this function within NX.

**Step 1:** Continue using **\*\*\*\_my\_post** in **Post Builder**.

- ☐ If necessary open **\*\*\*\_my\_post** using the **Post Builder**.

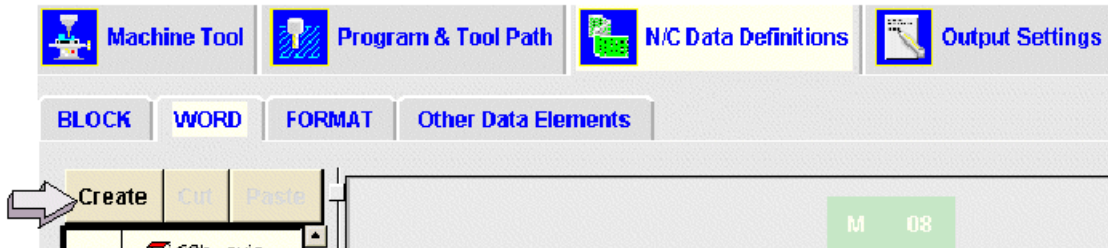
**Step 2:** Create the new M-code word group.

- ☐ Click **NC Data Definitions**.
- ☐ Click **Word**.

You will create the new M-code word group by copying an existing group.

**Step 3:** Scroll down and then highlight the **M\_coolant** word.

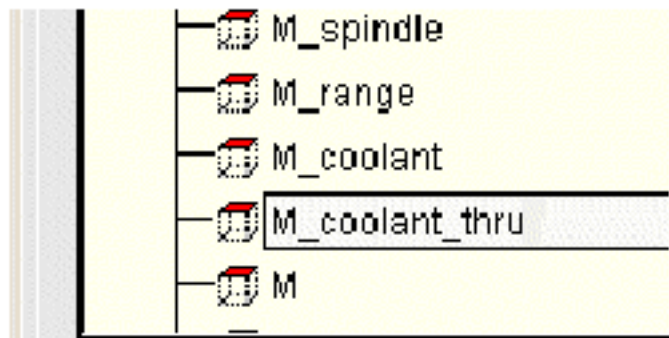
**Step 4:** Choose the **Create** button.



The **M\_coolant\_1** word is created.

**Step 5:** Change the name of the new word group.

- ☐ Click the **M\_coolant\_1** word.
- ☐ Change the name to **M\_coolant\_thru**.



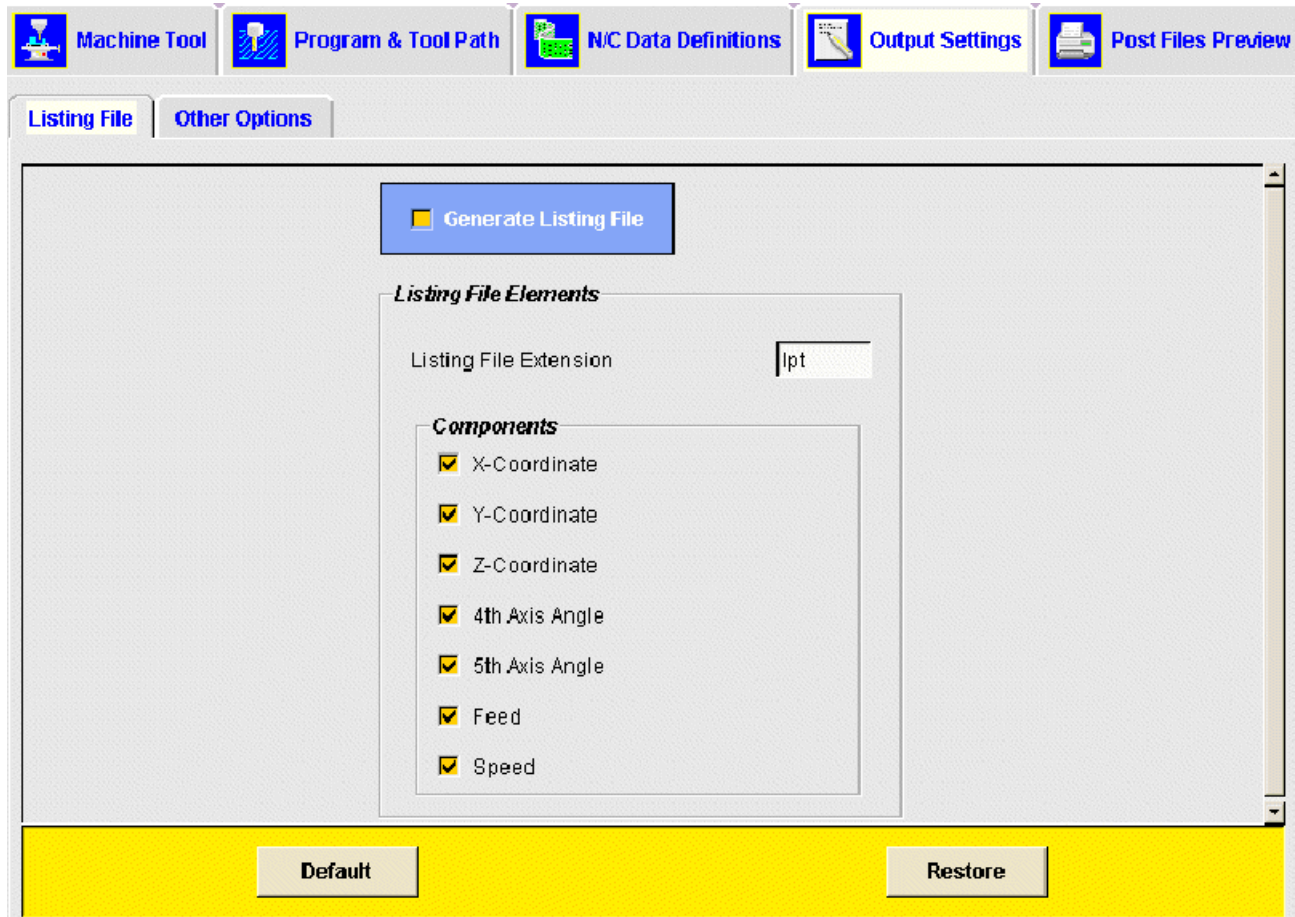
- ☐ Save the post processor.

This completes this activity.

In support of the creation of the new word **M\_coolant\_thru**, you will need to build a UDE to turn on the coolant through the tool, create a custom command to detect and interpret the coolant-through the tool UDE and then output the appropriate M-code. These steps will be performed in future activities.

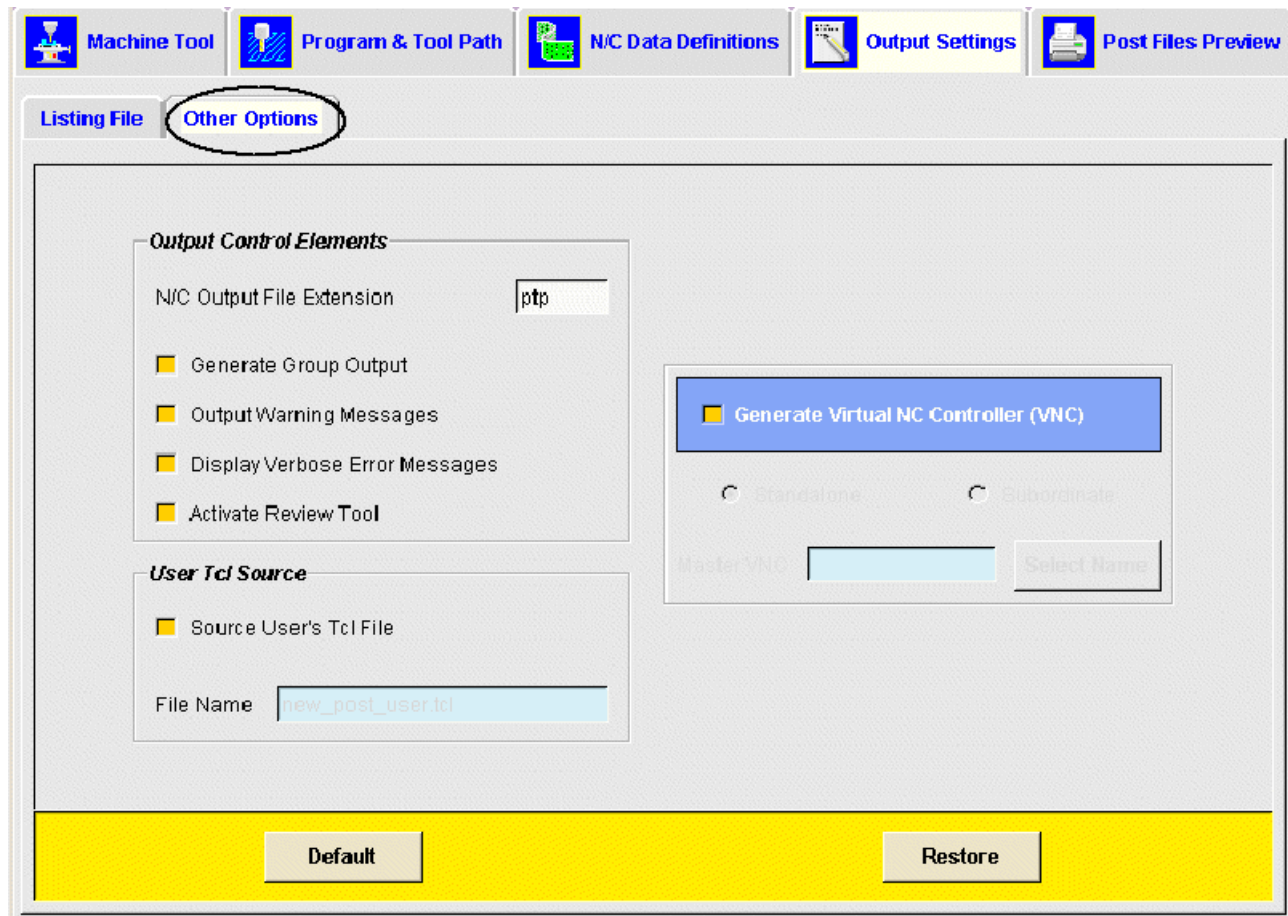
## Output Settings property page

The **Output Settings** property page allows the generation and control of output used in a listing file. Items for output are X, Y and Z co-ordinates, 4th and 5th-axis angle, and feeds and speeds. You can also specify file extensions for listing and NC output. Tabs are present for **Listing File** control and for **Other Options**.



Under **Other Options**, the **Output Control Elements** section, **Generate Group Output** allows the definition of how the post processor will process Groups. Operations may be organized into groups in order to create more than one NC program in a part file. Each group is considered to be a separate program. The default setting for this option is OFF. This option should only be used when you want to post process more than one NC program at a time. Selecting more than one Group with this option turned ON will result in one file containing all the Groups which have been selected. The file naming convention will be the name of your file with the name of the group appended. For example, if the base file name is "1234" and the group is called "finish\_mill", then the name of the file will be 1234\_finish\_mill.ptp. When Group output is set to ON, you should not mix operations in groups with operations outside of groups. This will result in unpredictable output. When the group output setting is OFF, you will only get a single NC output

file called 1234.ptp. All operations in any selected group will be output in a single file. Any group organization will be ignored.



2

The name of the group or program in NX can be used as the program name in the **Start of Program Event**. This will be passed as the variable name **mom\_group\_name**. By changing the name of the **PROGRAM** in the Operation Navigator, and by creating a custom command, you can create the equivalence of a PARTNO statement.

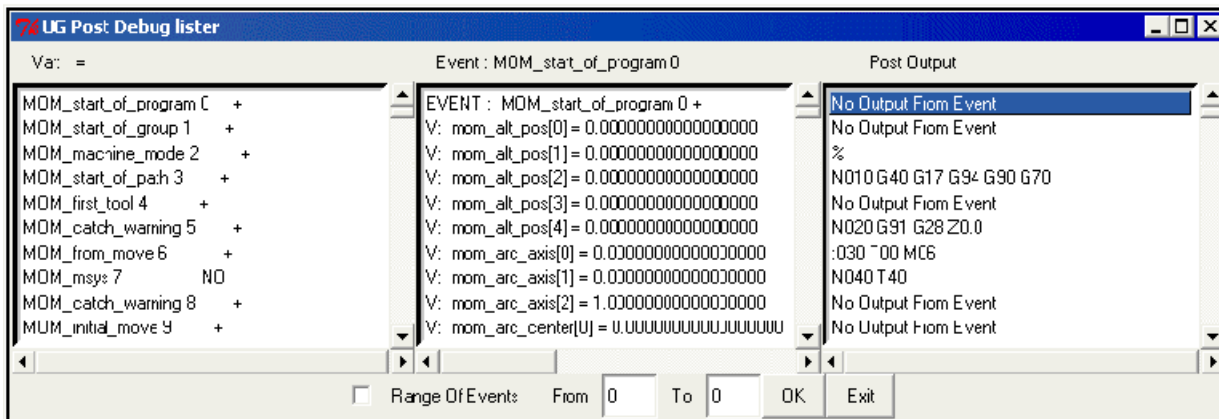
**Output Warning Messages** will result in warnings generated to a log file.

**Display Verbose Error Messages** will result in detailed error messages being displayed as your job is being post processed.

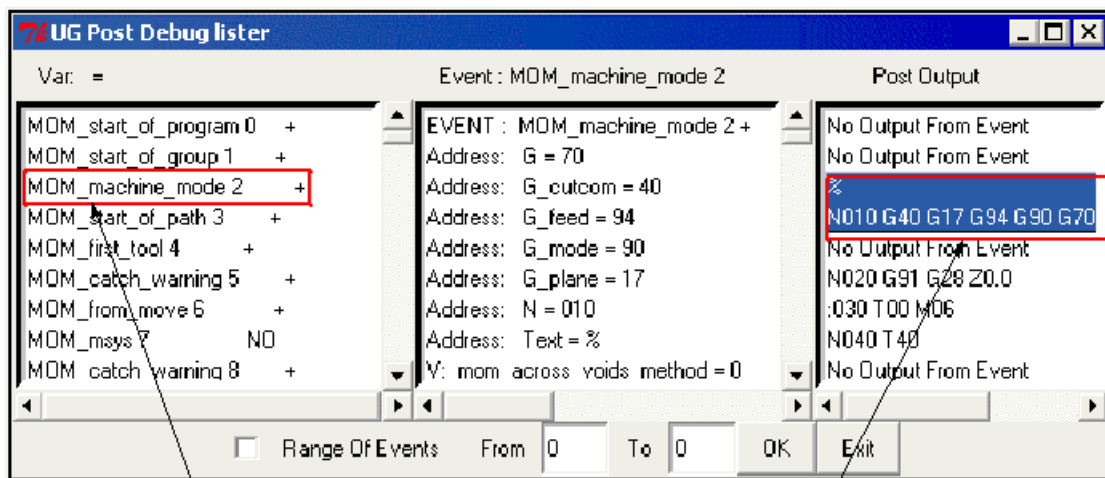
**Activate Review Tool** initiates the **Review Tool** which aides in the debugging of a post processor.

The **Review Tool** consists of three list boxes, each having horizontal and vertical scroll bars to assist in viewing the information.

2



The left most list box displays all the MOM events in sequential order that have been created during the post processing cycle. Each MOM event is displayed with the appropriate MOM event number. Selection of an event in this box will highlight the corresponding generated output code in the right list box.



Selecting a MOM event in this column highlights the corresponding output in the Post Output column

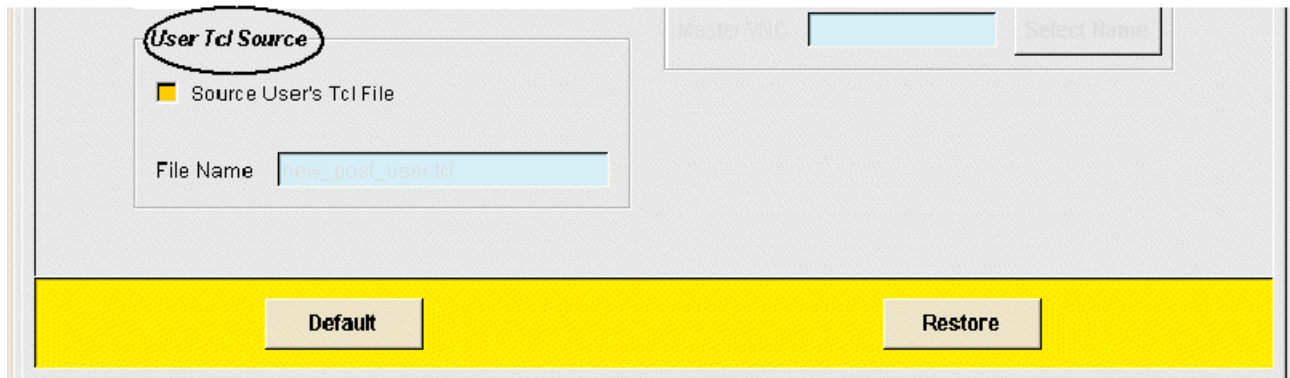
The middle list box displays the MOM generated variables and addresses associated with a particular Event in alphabetical order.

The right list box displays the output generated by the post processor.

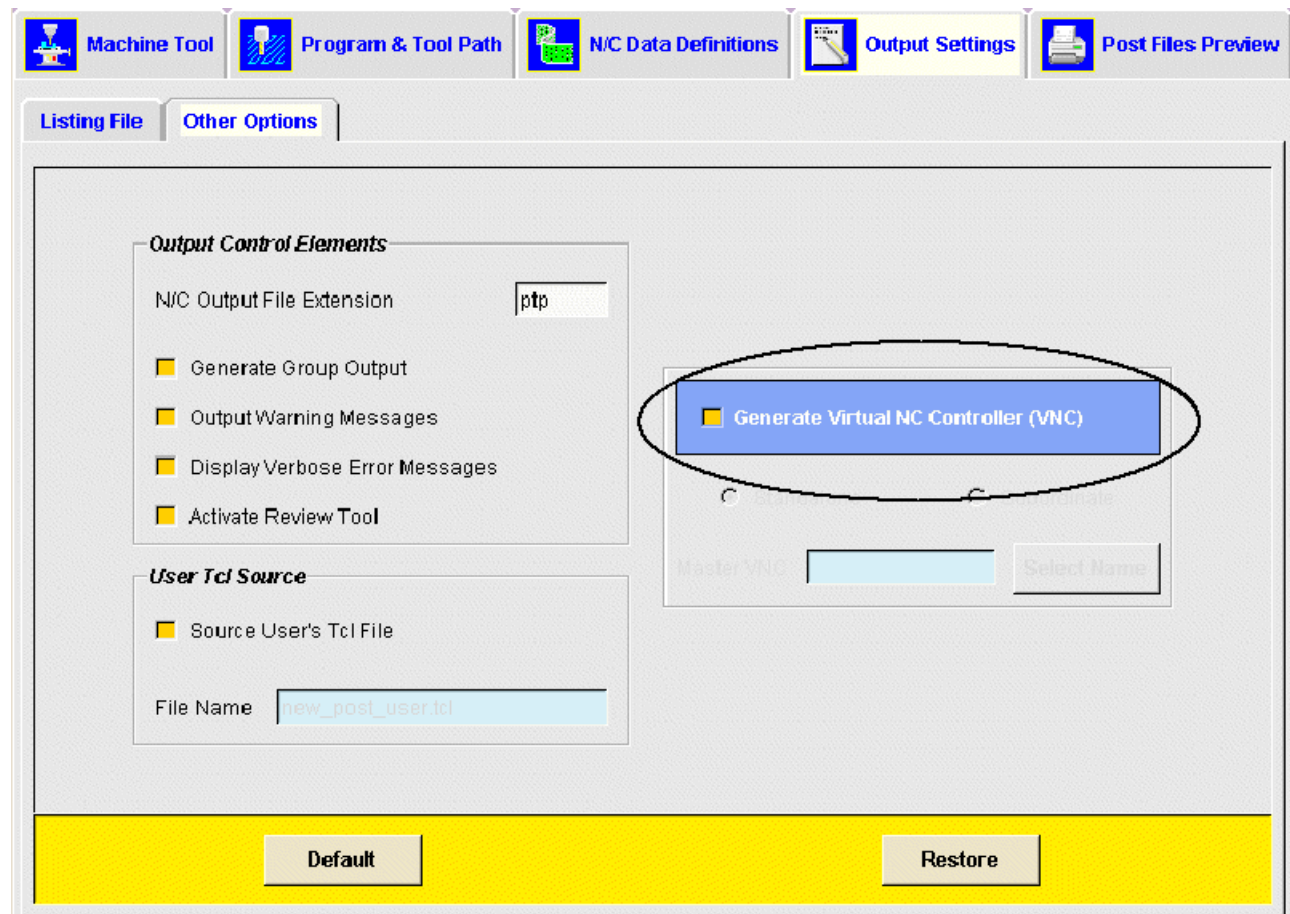
Note: Extensions defined inside of **Post Builder** will be superseded by the output file name displayed in the post processor dialog. Default extensions for listing output are **.lpt**, for machine output, the default is **.ptp**.

Under **User Tcl Source**, the **Source User's Tcl** file option allows the use of an existing Tcl source program.





The **Generate Virtual NC Controller (VNC)** option, when checked, creates a Virtual NC Controller which is used by the Integrated Simulation and Verification (IS&V) module of NX.



## Activity — Output settings

In this activity you will modify the data which is selected for output to a listing file.

**Step 1:** Enter the Output Settings section of **Post Builder**.

- ☐ Click **Output Settings**→**Listing File**.
- ☐ Select the **Generate Listing File** box.
- ☐ Clear the **4th and 5th axis angle** box.

**Step 2:** Save the post and run it against the test part, mill\_test.

- ☐ If necessary, enter the Manufacturing application in NX.
- ☐ Expand the **T12345-A** parent and select the **FACE\_MILLING** operation.



- ☐ Select **Post Process** .
- ☐ Select your post processor and select **OK**.
- ☐ Accept the default for **Output File** and then click **OK**.
- ☐ Verify the listing file. The listing file will be located in your home directory and will have a **.lpt** extension. Open the file with the Notepad editor and examine the contents.

The contents of the listing file should be similar to the following:

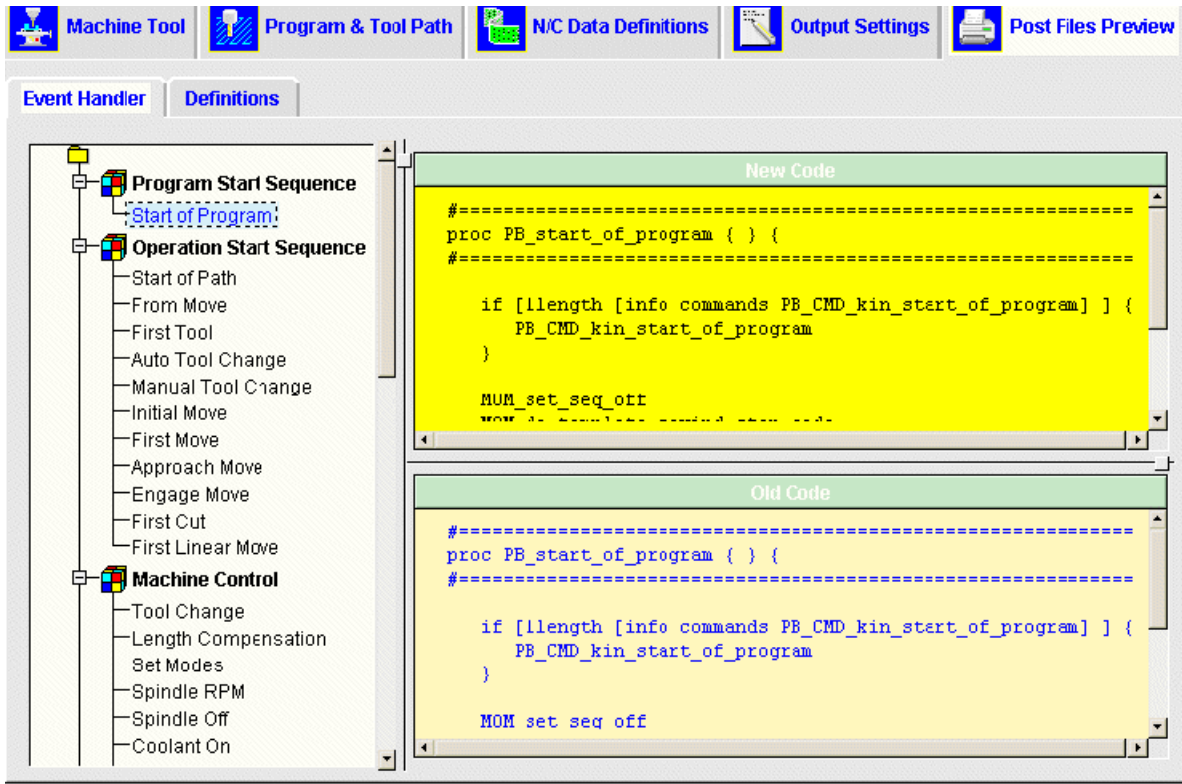


%	0.0000	0.0000	0.0000	0.0	0
N010 G40 G17 G94 G90 G70	0.0000	0.0000	0.0000	0.0	0
N020 G91 G28 Z0.0	0.0000	0.0000	0.0000	0.0	0
:030 T00 M06	0.0000	0.0000	0.0000	0.0	0
N040 T40	0.0000	0.0000	0.0000	0.0	0
N050 G00 G90 X-3. Y-3. S0 M03	-3.0000	-3.0000	6.0000	400.0	0
N060 G43 Z6.	-3.0000	-3.0000	6.0000	400.0	0
N070 X7. Y-3.1	7.0000	-3.1000	4.0000	400.0	0
N080 Z4.	7.0000	-3.1000	4.0000	400.0	0
N090 Z2.1	7.0000	-3.1000	2.1000	400.0	0
N100 G01 Z2. F13. M08	7.0000	-3.1000	2.0000	13.0	0
N110 Y-1.2	7.0000	-1.2000	2.0000	13.0	0
N120 X-1.2	-1.2000	-1.2000	2.0000	13.0	0
N130 Y9.2	-1.2000	9.2000	2.0000	13.0	0
N140 X15.2	15.2000	9.2000	2.0000	13.0	0
N150 Y-1.2	15.2000	-1.2000	2.0000	13.0	0
N160 X7.	7.0000	-1.2000	2.0000	13.0	0
N170 Y1.7766	7.0000	1.7766	2.0000	13.0	0
N180 X1.7766	1.7766	1.7766	2.0000	13.0	0
N190 Y6.2234	1.7766	6.2234	2.0000	13.0	0
N200 X12.2234	12.2234	6.2234	2.0000	13.0	0
N210 Y1.7766	12.2234	1.7766	2.0000	13.0	0
N220 X7.	7.0000	1.7766	2.0000	13.0	0
N230 Y3.5766	7.0000	3.5766	2.0000	13.0	0
N240 X3.5766	3.5766	3.5766	2.0000	13.0	0
N250 Y4.4234	3.5766	4.4234	2.0000	13.0	0
N260 X10.4234	10.4234	4.4234	2.0000	13.0	0
N270 Y3.5766	10.4234	3.5766	2.0000	13.0	0
N280 X7.	7.0000	3.5766	2.0000	13.0	0
N290 G00 Z4.	7.0000	3.5766	4.0000	400.0	0

☐ Return to the **Post Builder**.

This completes this activity.

## Post Files Preview property page



The **Post Files Preview** property page allows you to examine the Definition file (.def) and the Event Handlers (.tcl) before output generation. Newly defined code appears in the top window while the original code is displayed in the bottom window.

## Summary

The **Post Builder** is an efficient and robust tool that can be used to build the majority of the post processors that are required in today's manufacturing environment. The following functions were used in conjunction of building a post processor with the **Post Builder**:

- Created a directory structure for testing post processors developed with the **Post Builder**.
- Creation, modification and customization of Event handlers with the Program and Tool path function.
- Defined output formats.
- Modified listing and output options.
- Verified the output of your post processor using the **Post Review Tool**.



## Lesson

# 3 *Post Builder for Wire EDM applications*

3

### Purpose

This lesson describes the procedures for building post processors for Wire EDM applications through the use of the **Post Builder**.

### Objective

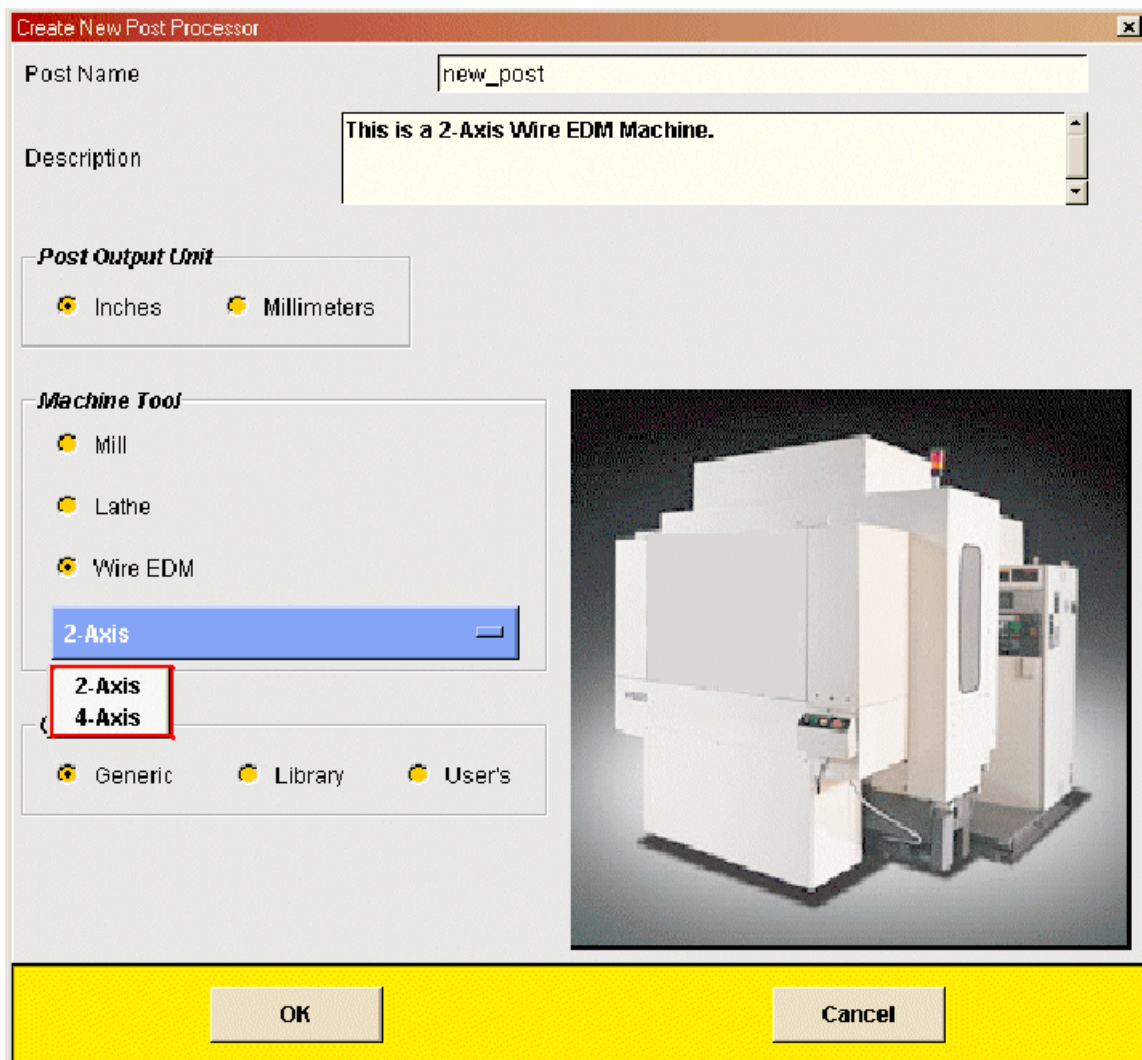
Upon completion of this lesson, you will be able to:

- Use the **Post Builder** to build 2 or 4-axis Wire EDM post processors.

## Use Post Builder to create 2-axis and 4-axis Wire EDM post processors

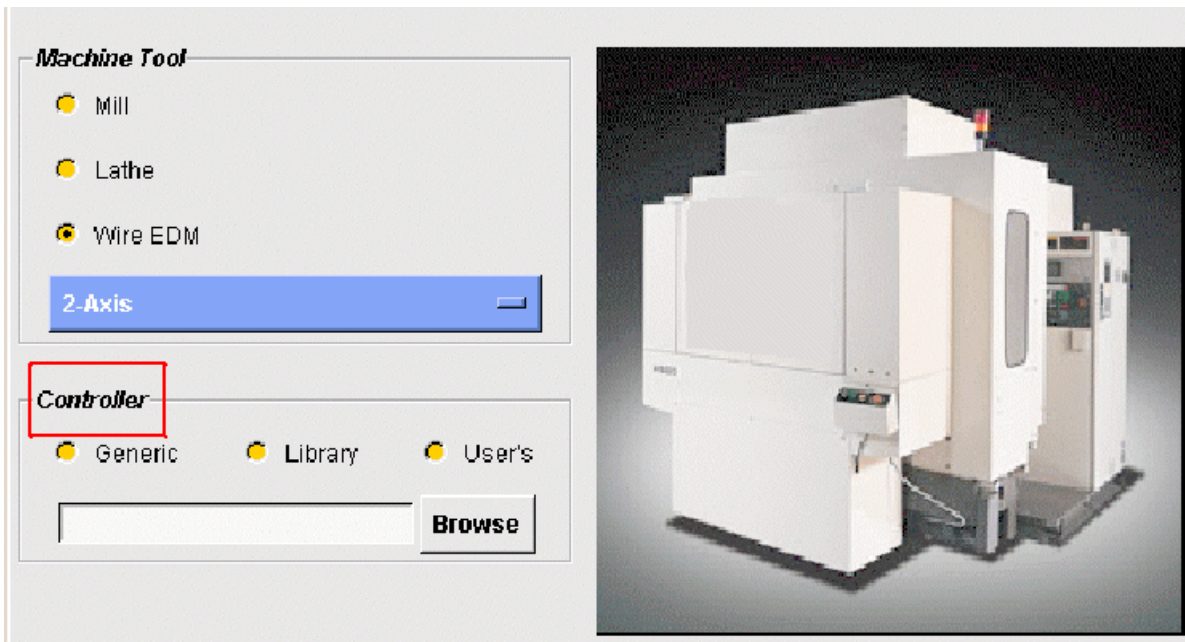
There are two different selections, available in the **Post Builder** for creating Wire EDM post processors. They are:

- 2-axis
- 4-axis



Once you make the selection of a 2-axis or 4-axis Wire EDM, you will then need to select the type of controller. Your choices available are:

- Generic
- Library
- User's

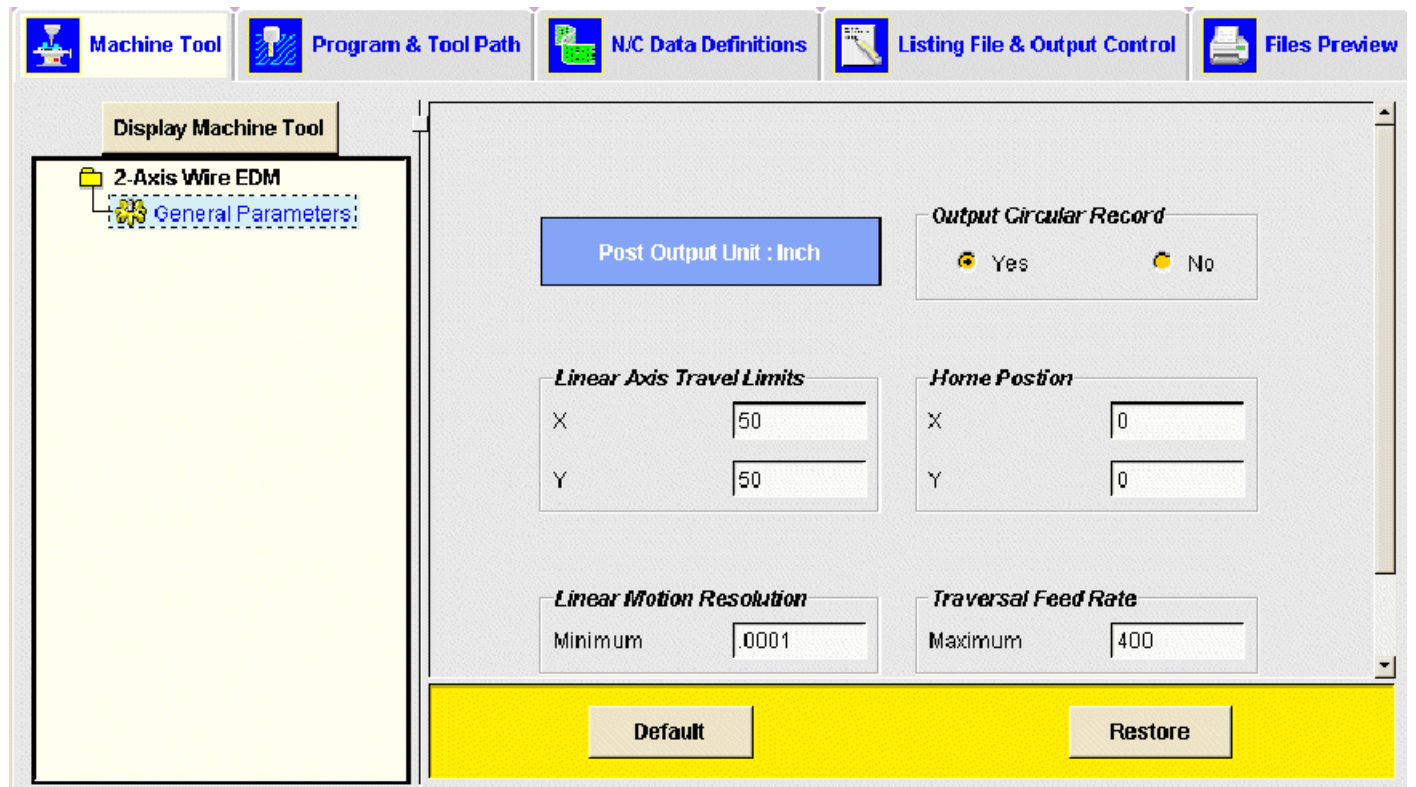


**Generic** controller defaults to a generic template that uses generic features common to most machine tool controllers.

**Library** controller allows the selection of templates for Agie, Charmilles and Mitsubishi (the main Wire EDM manufactures). These custom templates contain the basic codes required for the most common Wire EDM functions such as threading, start and end of cut and power settings required by the various controllers.

**User's** controller allows the selection, as a template, of a previous post that you have created. Selection is accomplished by choosing the **Browse** button and then selecting the previously created post processor from a specific directory.





The following activity will familiarize you with the functions and parameters that are available when creating a Wire EDM post processor.



## Activity — Create a 2-axis Wire EDM post

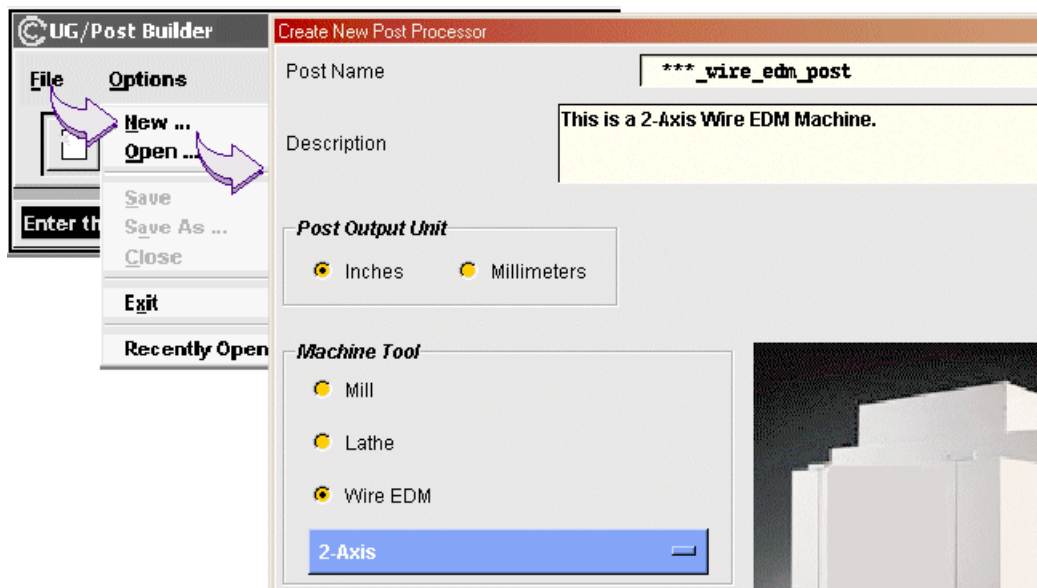
**Step 1:** If necessary, start the Post Builder.

- ☐ On the Windows desktop menu bar, choose **Start→All Programs→NX→Post Tools→Post Builder**

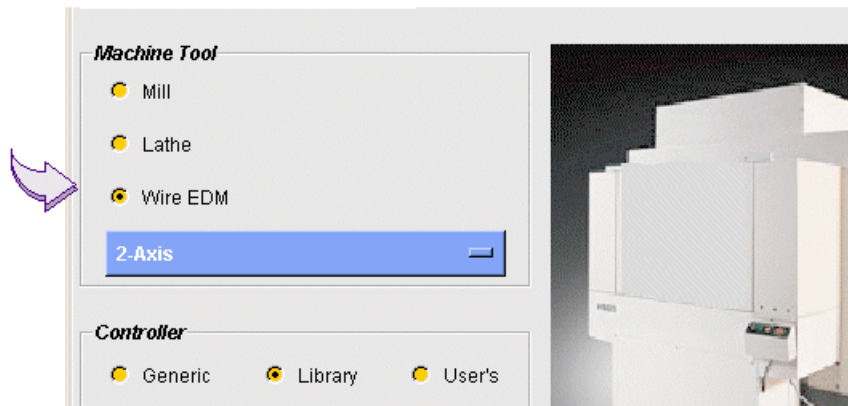
**Step 2:** Create a new 2-axis Wire EDM post.

- ☐ Select **New** in the **File** list and in the **Name** field, name the post processor **\*\*\*\_wire\_edm\_post**, where **\*\*\*** stands for your initials.

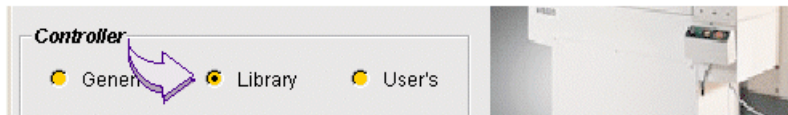
**Note:** Use lower case characters only and no spaces.



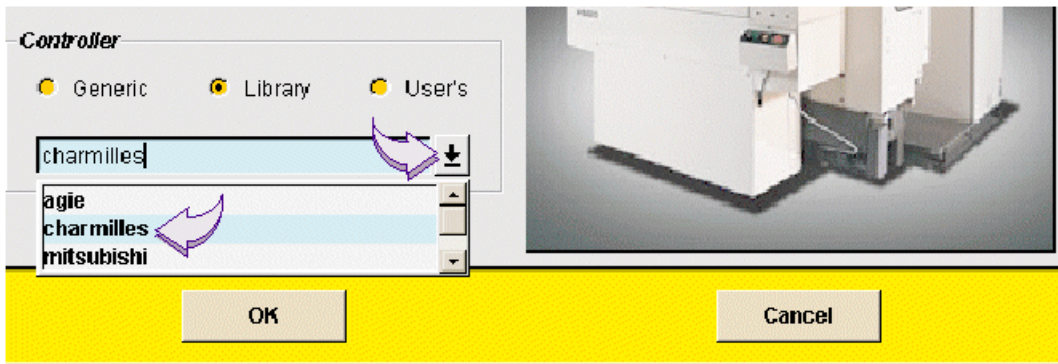
- ☐ Under **Machine Tool** , select **Wire EDM** and accept the default of 2-axis.



- ☐ Select **Library** from the **Controller** options.



- ☐ Select the down arrow and then select **Charmilles** from the pull down menu.



- ☐ Select **OK**.

You will accept all defaults to create the post, however, before you save your post, explore the various options available.

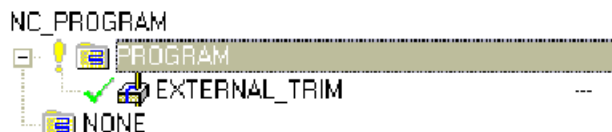
- ☐ Save the post to your home post processing directory by selecting **File, Save As** and then specifying your postprocessor location.

**Step 3:** You will open a part file and browse for the post processor.

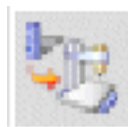
- ☐ Return to a NX session and retrieve the part file **wedm\_2\_4\_axis** from your **student\_home/parts** directory.

**Step 4:** Run the post against the test part, wedm\_2\_4\_axis.

- ☐ From the **Program View** of the Operation Navigator, select the **PROGRAM** parent.



- ☐ Select **Post Process** .



- ☐ Click **Browse** and select the **\*\*\*\_wire\_edm\_post** .

- ☐ Click **OK**.
- ☐ Click **OK** on the Postprocess dialog.

This completes this activity and the lesson.

## Summary

The flexibility and robustness of the Post Builder allows you to easily generate post processors for 2 and 4 axis Wire EDM machines. In this lesson you were introduced to:

- Creating a 2-axis Wire EDM post processor using the Post Builder tool.

## Lesson

# 4 *Post Builder for 5-Axis mill applications*

### Purpose

This lesson describes the procedures for building post processors for 5-axis mills through the use of the **Post Builder**.

### Objective

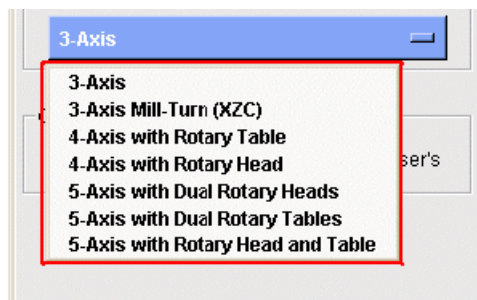
Upon completion of this lesson, you will be able to:

- Use the **Post Builder** to build 5-axis mill post processors.

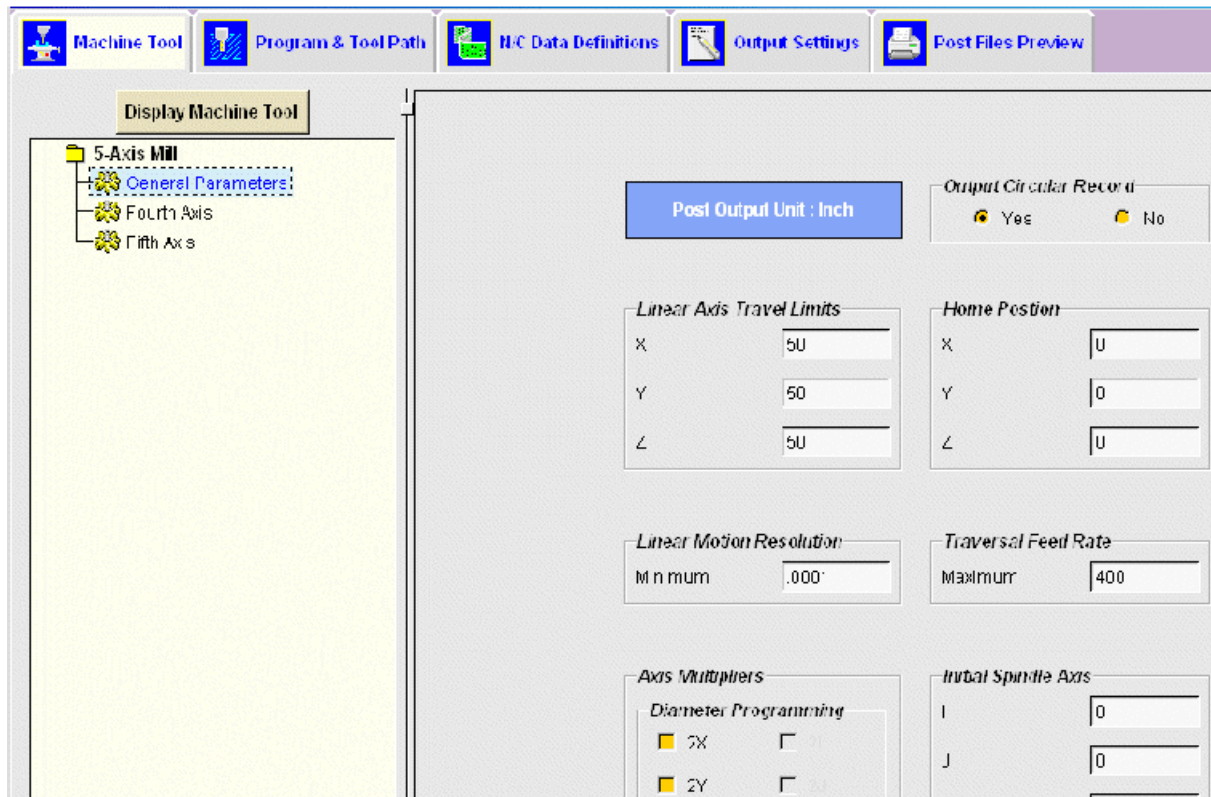
## Use Post Builder to create 5-axis Mill post processors

There are currently, five different selections, available in the Post Builder for creating 4 and 5-axis post processors. They are:

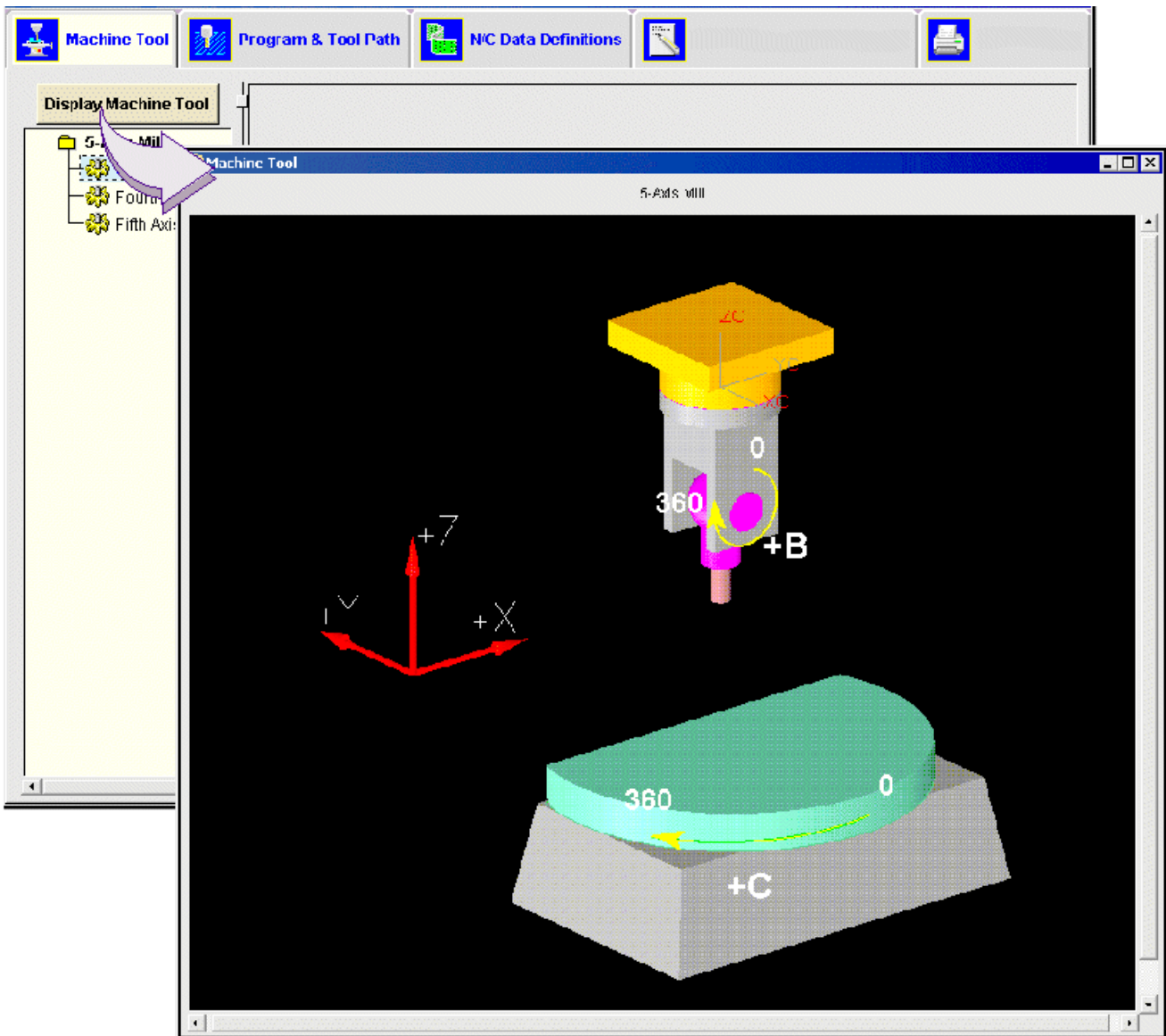
- 4-Axis with rotary table
- 4-Axis with rotary head
- 5-Axis with dual rotary heads
- 5-Axis with dual rotary tables
- 5-Axis with rotary head and table



Once you make the selection of the type of multi-axis machine, the **Machine Tool** property page is displayed, allowing you to choose **General**, **Fourth Axis** and **Fifth Axis** parameters from the component window of the property page.



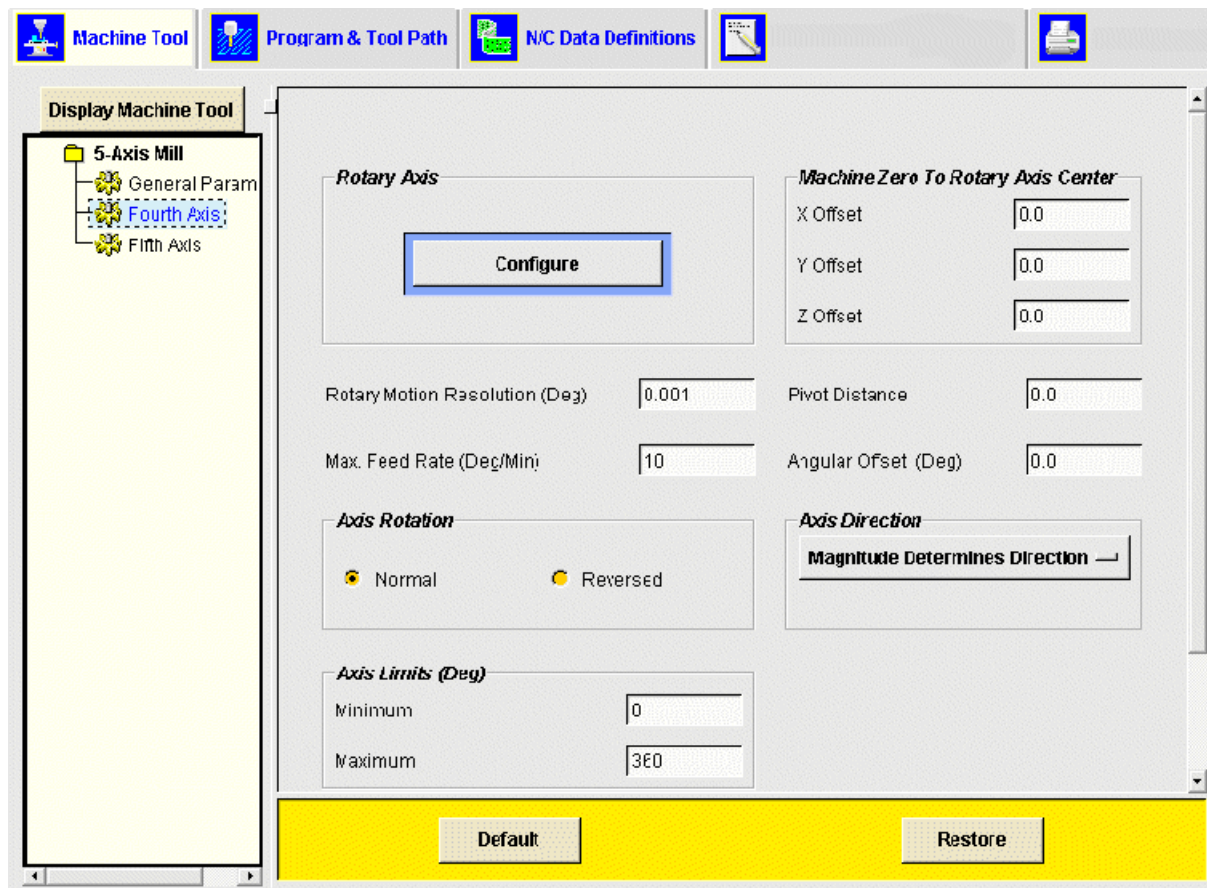
Selection of the **Display Machine Tool** button, displays a generic view of the machine tool that corresponds to the type of machine selected.



4

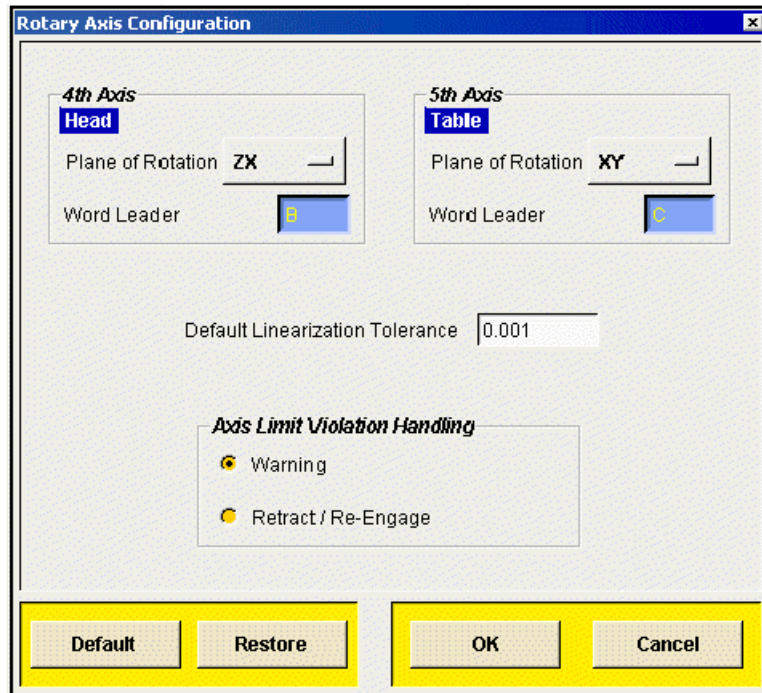


Selecting the 4th or 5th axis from the component window of the **Machine Tool** property page results in a rotary axis configuration property page being displayed. This property page allows you to configure the various parameters for each axis of rotary motion.



Parameters in this dialog are:

**Rotary Axis**- selection of the configure button results in the Rotary Axis Configuration dialog displayed. This dialog allows the selection of plane of rotation, axis word address designation, default linearization tolerance and the method of handling Axis Limit violations.



**Machine Zero to Rotary Axis Center** - allows the definition of the center of the 4th and or 5th axis in relation to the machine zero.

**Rotary Motion Resolution** - allows the specification of accuracy of rotation in degrees.

**Maximum Feed Rate** - allows the specification of the maximum degrees of rotation per minute.

**Pivot Distance** - specifies the distance from the head or table rotation pivot point to the spindle gage point. This option is not valid for dual rotary type machine tools.

**Angular Offset** - this offset adjusts the rotary angle motion and is generally used when the tool axis of 0,0,1 does not result in angular positions of 0 for the fourth and or fifth axis.

**Axis Rotation** - specifies the rotation of the 4th and or 5th axis according to the right-hand rule.

**Axis Direction** - determines clockwise or counterclockwise direction of rotary motion. Two options are available to determine axis direction. The first option, **Magnitude Determines Direction**, determines that the sign is used to determine the angular position. B-90 and B90 are different positions

and are 180 degrees apart. Rotation to a larger angle is always clockwise, to a smaller angle, counterclockwise. The second option, **Sign Determines Direction**, signifies that the sign only is used for direction. B-90 and B90 are the same position on the rotary table. B-90 designates that the table will rotate counterclockwise to B90. The reverse is true for B90 to B-90, the table will rotate clockwise.

**Axis Limits** - allows for control of the minimum and maximum angles that can be programmed for the fourth and or fifth axis. If the range of travel is limited to less than 360 degrees, the travel is measured clockwise. If there is no range of travel limits, there are two possible choices for minimum and maximum angles. If sign determines direction, 0 degrees is the minimum and 360 is the maximum. If magnitude determines direction, the minimum is -359.999 and the maximum is 359.999.

**Rotary Axis Can be Incremental** - this check box allows rotary output to be incremental with respect to the previous rotation.

## Activity – Create a 5–Axis Mill Post with Post Builder

**Step 1:** If necessary, start the Post Builder.

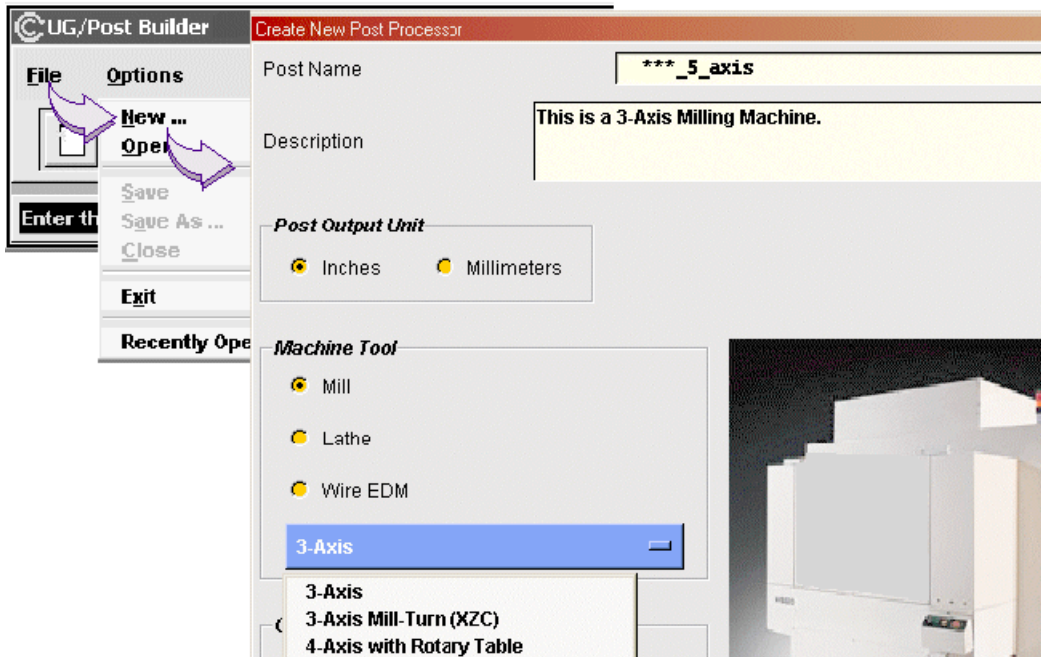
- ☐ On the Windows desktop menu bar, choose **Start→All Programs→NX→Post Tools→Post Builder**.

**Step 2:** Create a new 5-axis mill post with dual rotary tables.

- ☐ Select the **File→New** and in the **Name** field, name the post processor **\*\*\*\_5\_axis**, where **\*\*\*** stands for your initials.

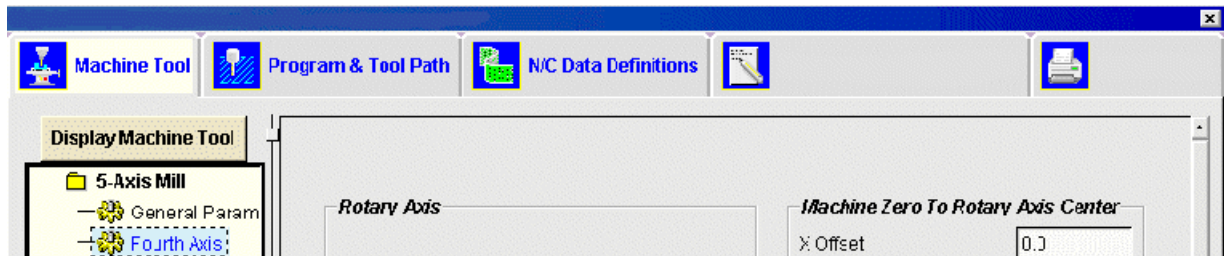
**Note:** Use lower case characters only and no spaces.

- ☐ Click **Inches** for the **Post Output Unit**.
- ☐ Click **Mill** for the **Machine Tool**.

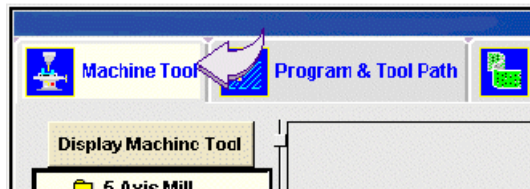


- ☐ Select **5-Axis with Dual Rotary Tables** from the **Machine Tool** list.
- ☐ Set the **Controller** to **Generic**.
- ☐ Click **OK**.

The Property Pages dialog is displayed.



- ☐ Click **Machine Tool**.



4

You will accept all defaults to create the post, however, before you save your post, explore the various options available under 4th and 5th axis located in the Component Window.

- ☐ Save the post by selecting **File**→**Save**.

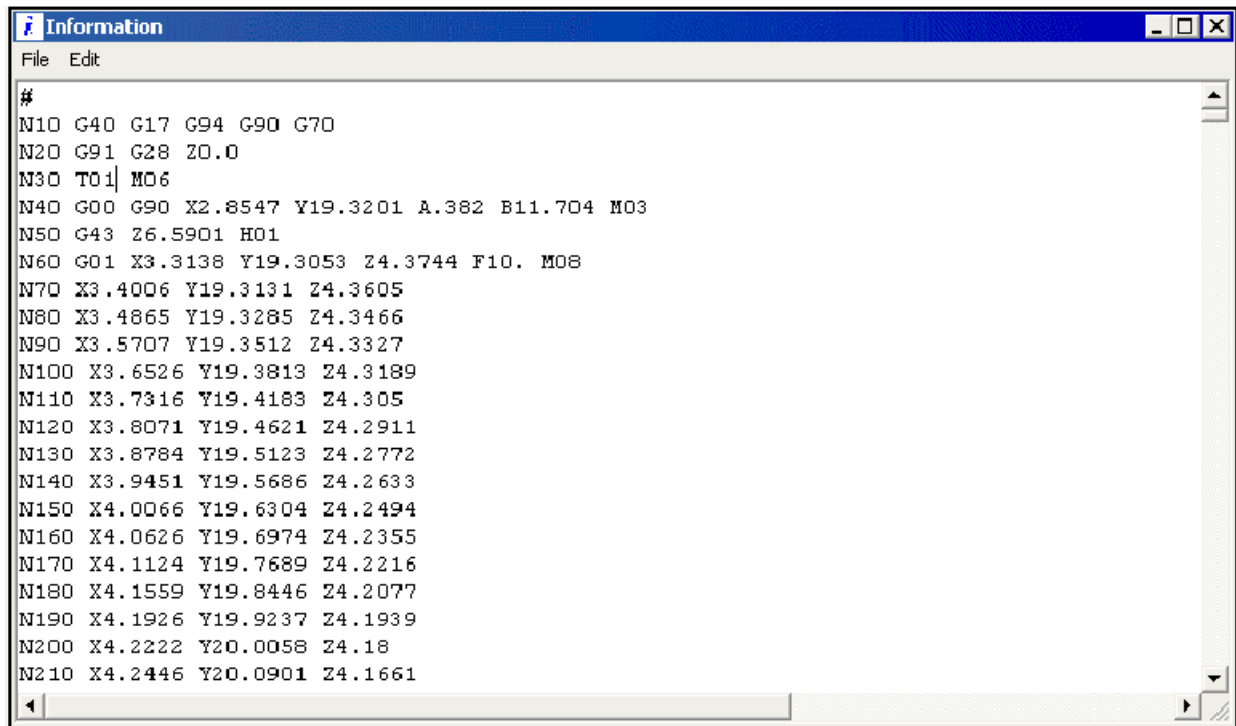
**Step 3:** Run the post against the test part, 5\_axis\_test.

- ☐ In NX, open the part file **5\_axis\_test** from your **student\_home/parts** directory.
- ☐ Select any operation from the list of operations in the operation navigator.



- ☐ Click **Post Process**.
- ☐ From the **Post Process** dialog in the **Available Machines** section of the dialog click **Browse**.
- ☐ Select **\*\*\*\_5\_axis** and choose **OK**.
- ☐ Choose **OK** to Post Process your operation.

Your output will be similar to the following:



```
#
N10 G40 G17 G94 G90 G70
N20 G91 G28 Z0.0
N30 T01 M06
N40 G00 G90 X2.8547 Y19.3201 A.382 B11.704 M03
N50 G43 Z6.5901 H01
N60 G01 X3.3138 Y19.3053 Z4.3744 F10. M08
N70 X3.4006 Y19.3131 Z4.3605
N80 X3.4865 Y19.3285 Z4.3466
N90 X3.5707 Y19.3512 Z4.3327
N100 X3.6526 Y19.3813 Z4.3189
N110 X3.7316 Y19.4183 Z4.305
N120 X3.8071 Y19.4621 Z4.2911
N130 X3.8784 Y19.5123 Z4.2772
N140 X3.9451 Y19.5686 Z4.2633
N150 X4.0066 Y19.6304 Z4.2494
N160 X4.0626 Y19.6974 Z4.2355
N170 X4.1124 Y19.7689 Z4.2216
N180 X4.1559 Y19.8446 Z4.2077
N190 X4.1926 Y19.9237 Z4.1939
N200 X4.2222 Y20.0058 Z4.18
N210 X4.2446 Y20.0901 Z4.1661
```

This completes this activity and the lesson.

The flexibility and robustness of the Post Builder allows you to easily generate post processors for 4 and 5 axis machining centers.

## Summary

In this lesson you were introduced to:

- Creating a 5-axis mill post processor using the Post Builder tool.





## Lesson

# 5 *Post Builder for lathe applications*

### Purpose

This lesson describes the procedures for building post processors for 2-axis lathes through the use of the **Post Builder**.

### Objective

Upon completion of this lesson, you will be able to:

- Use the **Post Builder** to build 2-axis lathe post processors.

## Use Post Builder to create Lathe post processors

There is currently, one selection, available in the Post Builder for creating lathe post processors. That selection is:

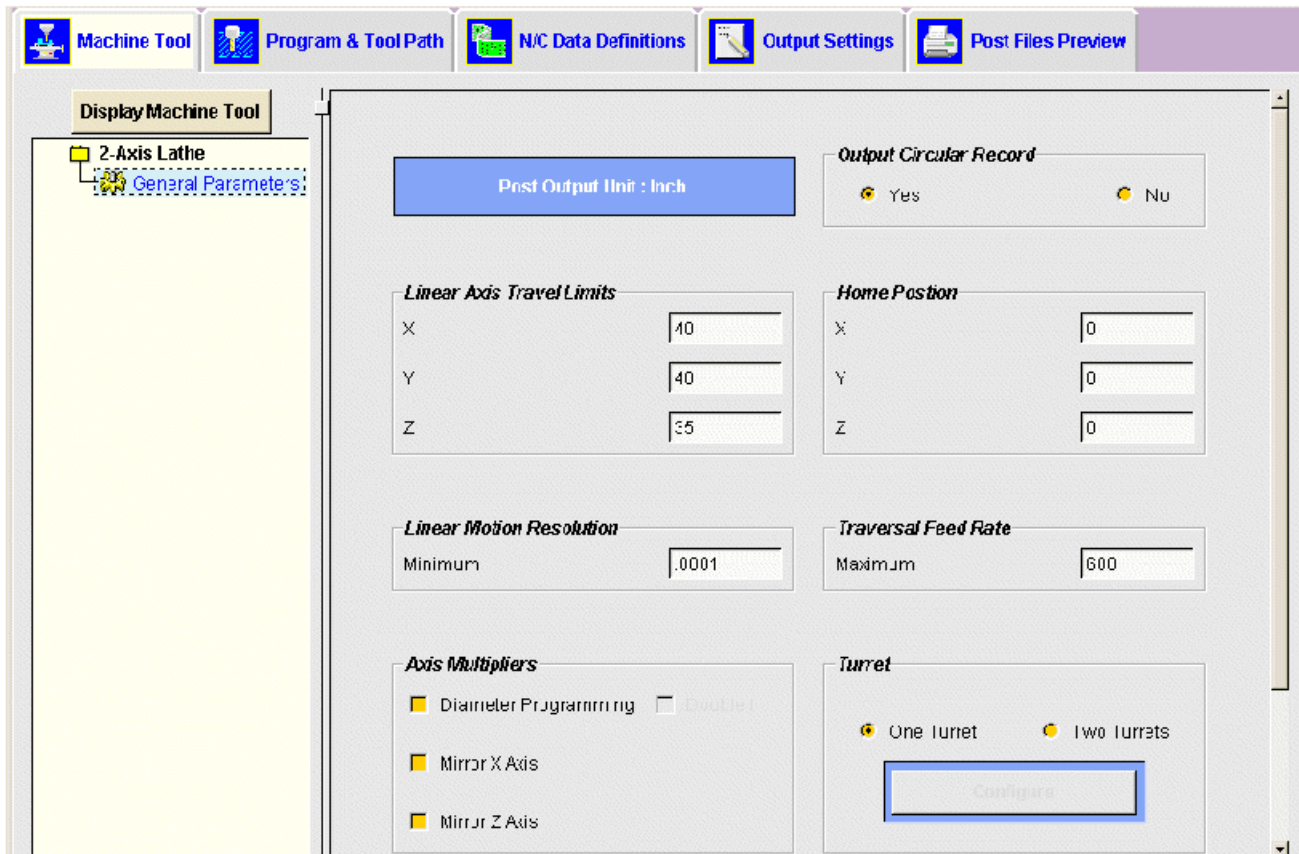
- 2-Axis

**Controller** allows the selection of **Generic**, **Library** or **User's** machine controllers. **Generic** controller contains defaults for a generic control. **Library** allows the selection of controller from a NX supplied list, currently for a Fanuc and Haas controller and **User's** allows the selection of post processors by **browsing** for specific post processors.

NOTE: You can select a previously created post processor.

The **Machine Tool** and **Controller** selections determine base files used to create the post processor which contains various Events, commands and procedures.

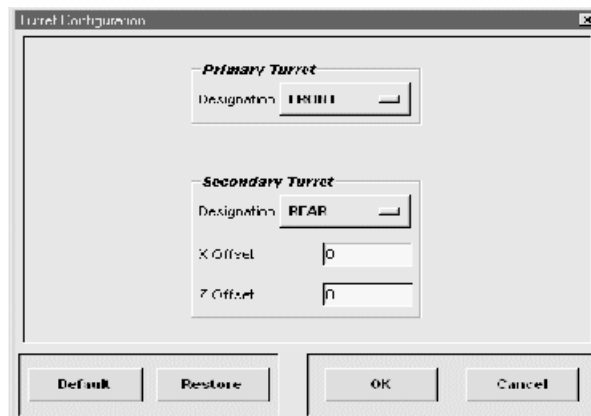
Once you make the selection of 2-axis lathe, general parameters for a lathe are available for selection from the Machine Tool property page.



5

The majority of the options/parameters available are self-explanatory. Those that need an expanded explanation are:

**Turret** - this option allows selection for machines having more than one turret. This option currently affects machines having turrets which do not move independently and are a fixed distance apart. When selecting this option, specify the names of the turrets, from a pull down list, and specify the distance between them. Currently six names are available.



**Output Method** - defines the basic tool tracking method. There are two options available with this method:

**Tool Tip** - X and Z values in the output file represent the tool tip location. With this option, the following is required:

- A FROM command at the beginning of the program to define the initial turret reference position.
  - A G92 block with each tool change to allow for changes in the offsets.
  - A GOHOME command to return back to a common turret reference position.

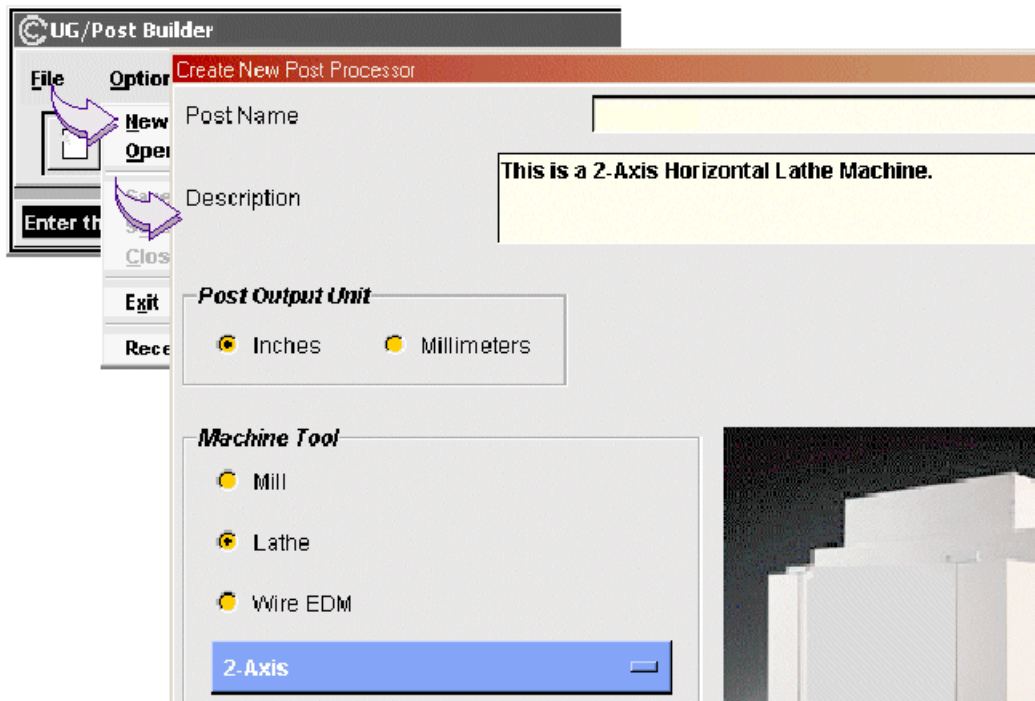
**Turret Reference** - X and Z values in the output file represent the turret reference location.

## Activity — Create a Lathe Post with Post Builder

**Step 1:** If necessary, start the Post Builder and create a new 2-axis lathe post processor.

- ☐ On the desktop menu bar, choose **Start→All Programs→NX→Post Tools→Post Builder**.
- ☐ Choose **File→New**, in the **Name** field type **\*\*\*2\_axis\_lathe**, where **\*\*\*** represents your initials.

**Note:** Use lower case characters only and no spaces.



The Create New Post Processor dialog is displayed.

- ☐ Under **Machine Tool**, select **lathe**.
- ☐ Choose **OK**.

The **Machine Tool** property page is displayed.

- ☐ Click **Machine Tool**.

You will accept all defaults to create the post, however, before you save your post, explore the various options available with lathe.

- ☐ **Save** the post by selecting **File→Save**.

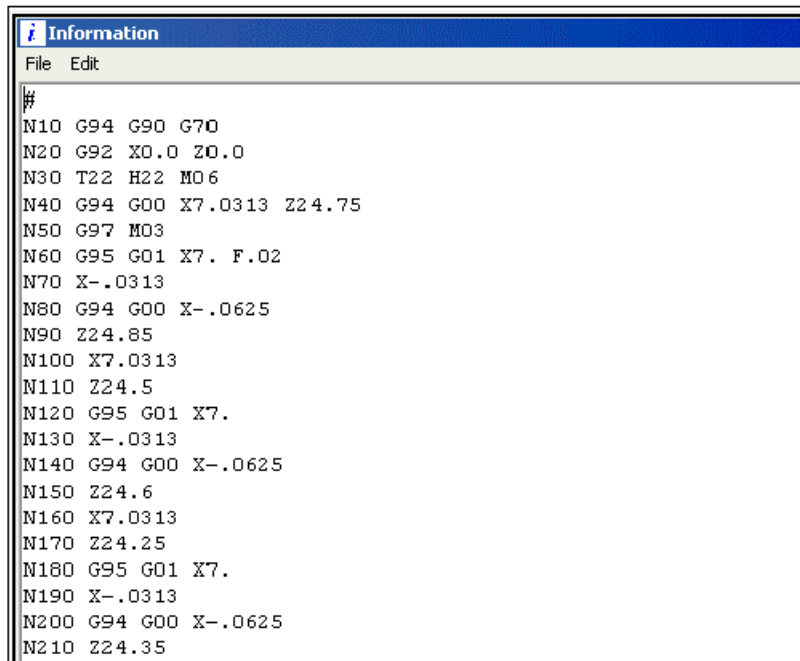
**Step 2:** Verify the post processor has been added to the Post Process dialog.

- ☐ Return to a NX session and retrieve the part file **lathe\_test.prt**.

**Step 3:** Run the post against the test part, lathe\_test.

- ☐ Expand and select the **P9876-A** program group object.
- ☐ Click **Post Process**.
- ☐ **Browse** for the **\*\*\*2\_axis\_lathe** post and choose **OK**.
- ☐ Select **\*\*\*2\_axis\_lathe** from the Post Process dialog.
- ☐ Choose **OK**.
- ☐ Verify the output.

The contents should be similar to the following:



```

#
N10 G94 G90 G70
N20 G92 X0.0 Z0.0
N30 T22 H22 M06
N40 G94 G00 X7.0313 Z24.75
N50 G97 M03
N60 G95 G01 X7. F.02
N70 X-.0313
N80 G94 G00 X-.0625
N90 Z24.85
N100 X7.0313
N110 Z24.5
N120 G95 G01 X7.
N130 X-.0313
N140 G94 G00 X-.0625
N150 Z24.6
N160 X7.0313
N170 Z24.25
N180 G95 G01 X7.
N190 X-.0313
N200 G94 G00 X-.0625
N210 Z24.35
  
```

Return to the Post Builder.

This completes this activity and the lesson.

## Summary

The flexibility and robustness of the Post Builder allows you to easily generate post processors for 2-axis lathes. In this lesson you were introduced to:

- Creating a 2-axis lathe post processor using the Post Builder tool.





## Lesson

# 6 *Create Mill-Turn post processors*

### Purpose

This lesson describes the procedures required to build Mill-Turn post processors through the use of **the Post Builder**.

### Objective

Upon completion of this lesson, you will be able to:

- Construct post processors used in mill-turn applications.

## Mill-Turn centers

The basic mill-turn center, along with turning tools, uses a rotary cutting tool that is mounted in one or more pockets on the mill-turn center's turret. Various attachments, including right angle heads, allow cutting along the X-axis while standard heads cut along the Z-axis of travel. On these types of machines, any milling or drilling from the machine's center line is not feasible because of the two-axis limitation of the turning center. The rotary C-axis is normally an orientation axis; rotation is normally inhibited when milling, drilling or tapping.

More complex mill-turn centers have contouring and positioning control of the rotary C-axis, which allows for complex spiral and contour milling cuts. Many of these types of machines also have an additional spindle, mounted on its own set of slides, which gives the ability to machine the opposite side of the part in one set-up.



Typical mill-turn application

An easy approach to programming such machines would be to segregate and then post process the various turning and milling (drilling, as well) operations. A post processor would be created for the turning functions, a separate post would be created for the drilling and milling functions.

A mill-turn post processor can be considered to be a separate mill and lathe post which are linked together. When executed, this linked post will check the machine mode and either loads the lathe or the mill post.

## Mill-Turn and the Post Builder

Creating a mill-turn post using the Post Builder is relatively simple. Use the following as a guideline:

- Create a new 2-axis lathe post, save the post and then close the file
- Create a new mill post, selecting the mill-turn option (XZC)
- Verify that you have used the **MCS\_SPINDLE** group object for all lathe tool paths
- Verify that you have used the **MCS\_MILL** group object for all milling tool paths
- Post process your tool paths

## Heads for Mill-turn centers

Some of the more sophisticated mill-turn centers use special **Heads** that are used for turning, z-axis, x-axis, y-axis or 5-axis work. These heads can be utilized by the HEAD User Defined Event:

## Create a Mill-Turn Post Processor

**Step 1:** Start the Post Builder.

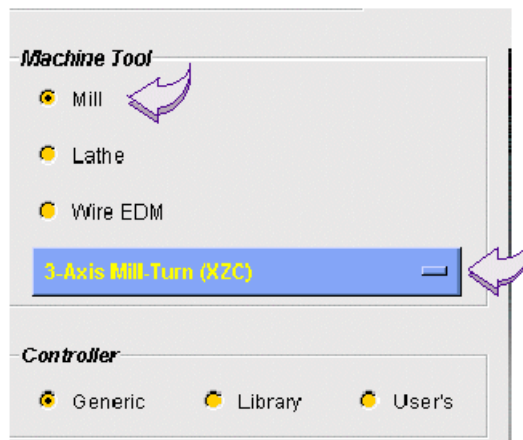
- ☐ Click **Start**→**All Programs**→**NX**→**Post Tools**→**Post Builder**.

**Step 2:** Create a new 2-axis lathe post processor.

- ☐ Click **File**→**New**, type **\*\*\*\_mill\_turn\_lathe** in the **Name** box, where **\*\*\*** represents your initials.
- ☐ Select the **Lathe** box for the **Machine Tool**.
- ☐ Click **OK**.
- ☐ Save your post processor in your home post processor directory.
- ☐ Click **File**→**Close** to close the post.

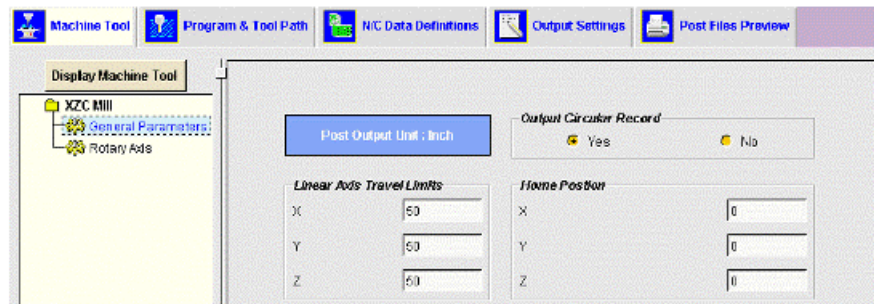
**Step 3:** Create a mill-turn post processor.

- ☐ Click **New** , in the **Name** box type **\*\*\*\_mill\_turn**, where **\*\*\*** represents your initials.
- ☐ Select the **Mill** check box for the **Machine Tool** and select the **3-Axis Mill-Turn (XZC)** from the list.



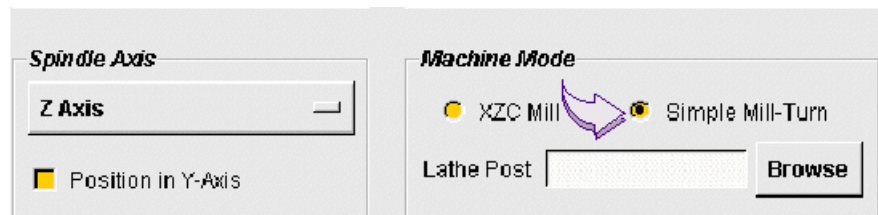
- ☐ Click **OK**.

The Machine Tool property page is displayed.

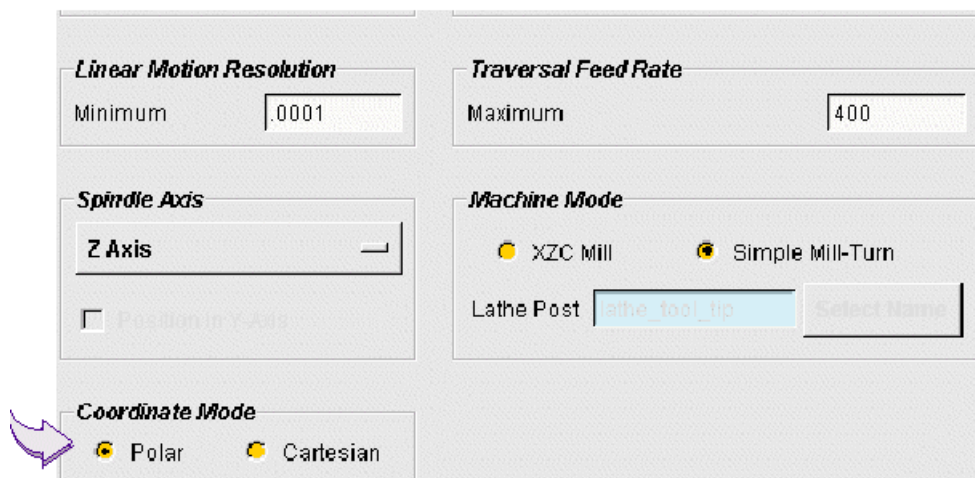


By default, the mill post of the mill-turn is the current post that you are creating. The lathe post defaults to the **lathe\_tool\_tip** post in the post processor directory. If you have a special lathe post that you would like to use, you would browse and then select that post. You will accept the current defaults.

- ☐ In the Machine Mode area of the dialog, select the **Simple Mill-Turn** box..



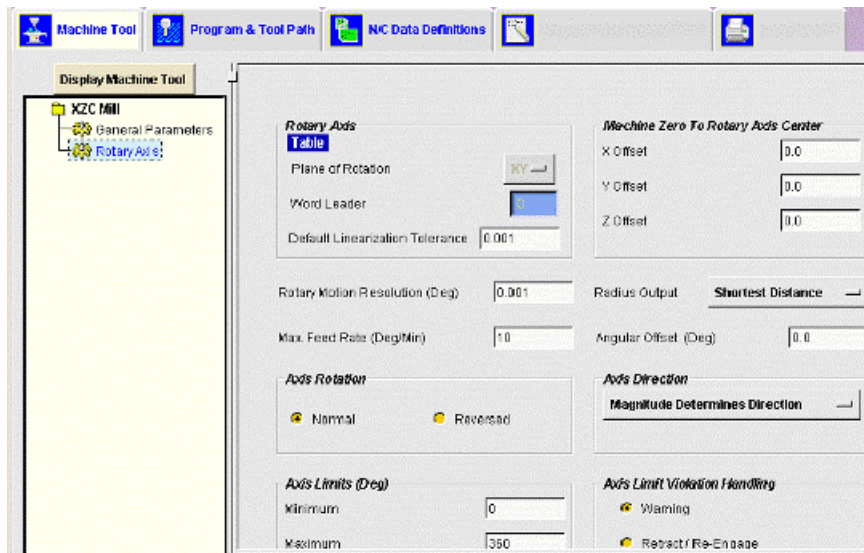
- ☐ Click **Browse**.
- ☐ Select the **\*\*\*\_mill\_turn\_lathe** post processor created. in step 2.
- ☐ Select the **Polar Coordinate Mode** box.



- ☐ Select **Rotary Axis** from the Machine Tool page.



- ☐ Examine the various parameters which are available. You will accept the defaults.



- ☐ Click the **Program & Tool Path** tab.
- ☐ Click **Word Sequencing**.
- ☐ Click '**Y**' and '**J**' addresses, if not already suppressed.

**Step 4:** Save your newly created post processor to your home directory.

- ☐ Select the **File** pull down menu from the main menu bar and then select **Save As**.
- ☐ Filter to your home post processor directory.
- ☐ Visually check the **File name** box, if the file name shown is the same as the file name of your post processor, select **OK**, otherwise type in the correct name and then select **OK**.

You have just created the mill post processor required for the mill-turn center. You will now test your post processor to verify the output.

**Step 5:** Verify the output.

- ☐ Return to a NX session and retrieve the part file **millturn\_mfg** from your **student\_home/parts** directory.
- ☐ Choose **Start→Manufacturing**
- ☐ Click **NC\_PROGRAM** from the Program View of the Operation Navigator.



- ☐ Select the **Post Process** .
- ☐ Select **Browse** and select **\*\*\*\_mill\_turn**.
- ☐ Click **OK**.
- ☐ Click **OK** on the Postprocess dialog.
- ☐ Verify the output.



Your output will be similar to the following:

Turning Output	Milling Output
N0010 G40 G17 G94 G90 G70 <sub>2</sub>	N2970 G94 G00 X4.5398 <sub>2</sub>
N20 G92 X0.0 Z0.0 <sub>2</sub>	N2980 X5.5 <sub>2</sub>
N30 T01 H01 M06 <sub>2</sub>	N2990 Z12.9 <sub>2</sub>
N40 G97 S0 M03 <sub>2</sub>	N3000 G91 G28 Z0.0 <sub>2</sub>
N50 G94 G00 G90 X5.6 Z13.2 <sub>2</sub>	:3010 T08 M06 <sub>2</sub>
N60 X5.6938 Z12.7024 <sub>2</sub>	N3020 T25 <sub>2</sub>
N70 G92 S0 <sub>2</sub>	N3030 G12.1 <sub>2</sub>
N80 G96 S356 M03 <sub>2</sub>	N3040 G00 G90 X-7.0711 Y0.0
N90 G95 G01 X5.6469 F03 <sub>2</sub>	C-8.13 S2712 M03 <sub>2</sub>
N100 X0.0 <sub>2</sub>	N3050 G43 Z14.4 H08 <sub>2</sub>
N110 X-.0469 F05 <sub>2</sub>	N3060 X-4.2759 C3.352 <sub>2</sub>
N120 G94 G00 Z12.8024 <sub>2</sub>	N3070 Z12.9 <sub>2</sub>
N130 X5.6938 <sub>2</sub>	N3080 Z11.5 <sub>2</sub>
N140 Z12.4579 <sub>2</sub>	N3090 G94 G01 Z11.4 F7 M08 <sub>2</sub>
N150 G95 G01 X5.6469 F03 <sub>2</sub>	N3100 X-4.1128 C3.485 <sub>2</sub>
N160 X0.0 <sub>2</sub>	N3110 X-3.9765 C3.605 <sub>2</sub>
N170 X-.0469 F05 <sub>2</sub>	N3120 X-3.8245 C3.748 F6.2 <sub>2</sub>
N180 G94 G00 Z12.5579 <sub>2</sub>	N3130 X-3.6781 C3.897 <sub>2</sub>
N190 X5.6938 <sub>2</sub>	N4650 G84 X3. Y0.0 Z10.4557
	R11.5 F3.3 <sub>2</sub>
	N4660 X-3.2 <sub>2</sub>
	N4670 G80 <sub>2</sub>
	N4680 G00 Z12.9 <sub>2</sub>
	N4690 Z15.9 <sub>2</sub>
	N4700 X0.0 <sub>2</sub>
	N4710 M02 <sub>2</sub>
	% <sub>2</sub>

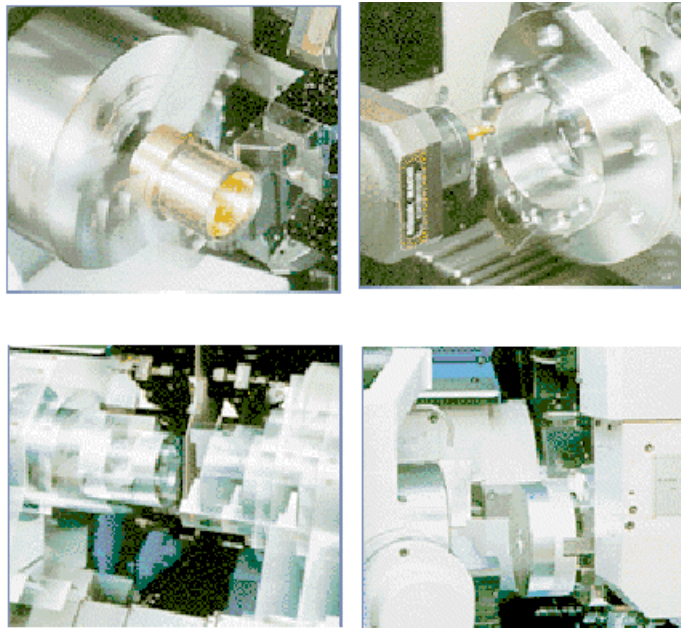
☐ **Save** and **Close** the current post processor.

This completes this activity.

## 5 Axis Mill-Turn centers

The combination of heavy duty turning centers with a high performance machining center on one base has resulted in a 5-axis mill-turn center with X, Z, C, B and Y axes movement.

These machines are generally configured where the spindle head holds the chuck for C-axis turning, the turret is mounted on a slide, providing for X and Z-axis movement, while synchronous turret movement provides travel in the Y axis. The turret has B-axis movement. The addition of Y-axis turret travel provides for off-center drilling and tapping as well as cross-cut milling. The B-axes movement of the turret also allows the machining of inclined surfaces and oblique holes as well as precision contour and helical milling. Contouring and positioning control of the rotary C-axis, allows for complex spiral and contour milling cuts. Many of these types of machines also have an additional spindle, mounted on its own set of slides, which gives the ability to machine the opposite side of a part in one set-up.



The same approach used for programming simple mill-turn centers would also be used for 5-axis applications (segregate and then post process the various turning, milling and drilling, operations). A post processor would be created for the turning functions, a separate post would be created for the drilling and milling functions.

## 5-Axis Mill-Turns and the Post Builder

Creating a 5-axis multi-linked mill-turn post using the Post Builder is relatively simple using the following guidelines:

- Create and save a new 2-axis lathe post
- Create a new 5-axis mill post, with rotary head and rotary table
- Create a new mill-turn XZC post which calls all other posts; use the 3-axis mill turn option.
- Create the necessary methods for turning operations, z-head mill operations, z-head drill operations, x-head mill operations and x-head drill operations
- Move all lathe, mill and drill operations to their suitable method parent
- Create and assign Head UDE's to each of the newly created method parents
- Post process your tool paths

The following activity will take you through the process of creating and using a 5-axis multi-linked mill turning post processor.

## Activity — Create a 5-Axis multi-link mill-turn post

In this activity, you will create a 5-axis multi-linked mill-turn post. This particular post is representative of a mill-turn with one spindle (C-Axis) and two tooling heads, one in the X-Axis and one in the Z-Axis. The X-Axis also is capable of B-Axis rotation. Three post processors are required and are linked together:

- A turning post, used for all turning utilizing the front spindle
- A post for the XHEAD, used for all milling and drilling in the X-axis
- A post for the ZHEAD, used for all milling and drilling in the Z-axis

You must also assign the proper UDE to each Method parent.

You will first create the necessary post processors and then modify existing operations to coincide with the functionality of the post processors.

**Step 1:** If necessary, start the Post Builder.

- ☐ Click **Start→All Programs→NX→Post Tools→Post Builder**.

**Step 2:** Create a 2-axis lathe post processor.

- ☐ Click **File→New**, select the **Lathe** check box, and in the **Name** field, type **\*\*\*\_mill\_turn\_5ax\_turn**, where **\*\*\*** represents your initials.
- ☐ Select **OK**.

Save and Close the lathe post.

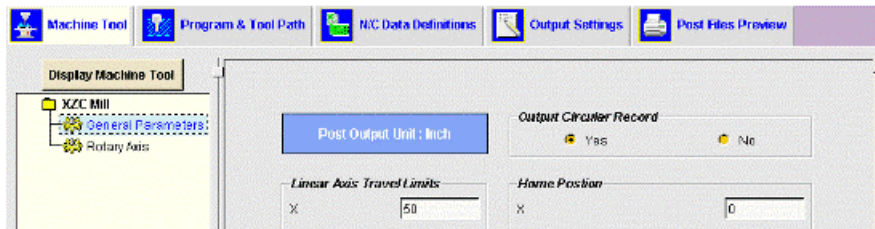
- ☐ File **File→Save**.
- ☐ Choose **File→Close..**

You will now create the necessary mill post.

**Step 3:** Use Post Builder to create a new mill 5-axis head table post processor for the X-Axis head.

- ☐ Choose **File→New**, select the **Mill** check box, select the **5-Axis with Rotary Head and Table** from the list and in the **Name** field, type **\*\*\*\_mill\_turn\_5ax\_xhead**, where **\*\*\*** represents your initials.
- ☐ Click **OK**.

The Machine Tool property page is displayed.

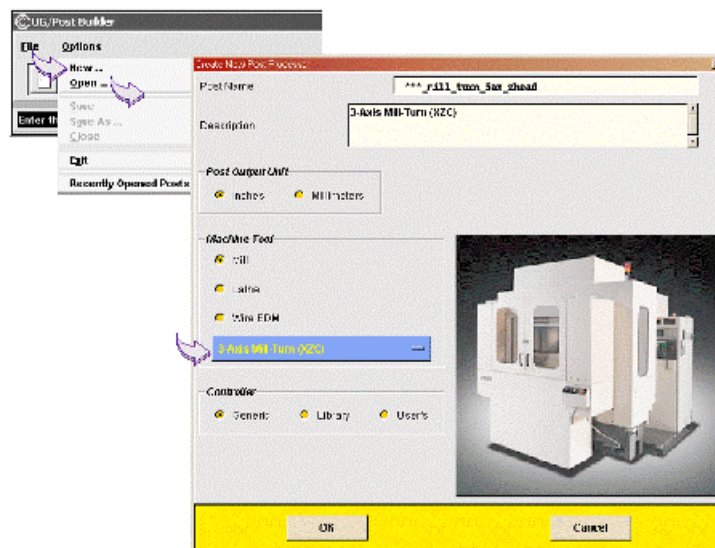


- ☐ Select the **Program & Tool Path**→**Program**→**Canned Cycles** component and verify the fourth (B) and fifth (C) axis addresses appear in the **Common** parameters block.
- ☐ Select **OK**.
- ☐ **Save** and **Close** the **\*\*\*\_mill\_turn\_5ax\_xhead** post.

You will now create the post processor for the Z-Axis head.

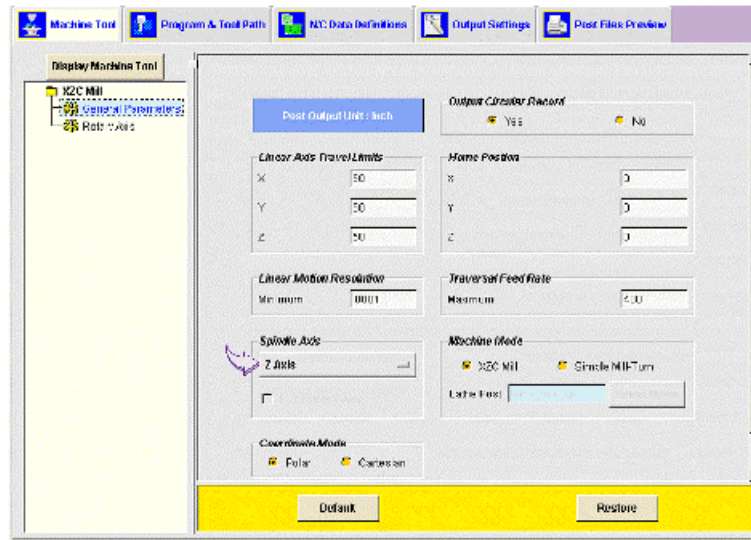
**Step 4:** Use Post Builder to create a new mill 5-axis head table post processor for the Z-Axis head.

- ☐ Click **File**→**New**, select the Mill check box, then select **3-Axis Mill-Turn (XZC)** from the list and in the **Name** field, type **\*\*\*\_mill\_turn\_5ax\_zhead**, where \*\*\* represents your initials. This is the main post that will call all others.



- ☐ Click **OK**.

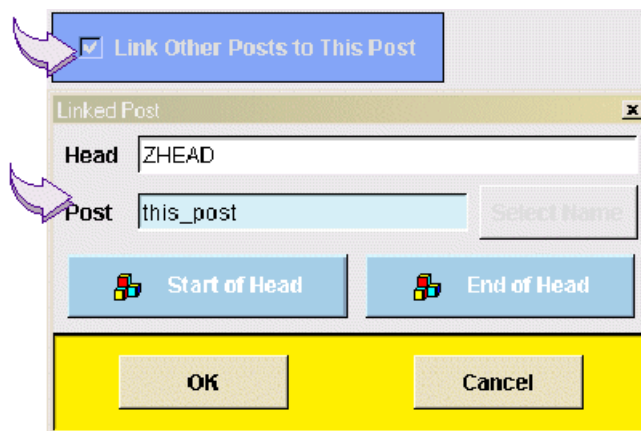
- On the **Machine** property page, **General Parameters**, verify the **Spindle Axis** is set to **Z-Axis**.



- Choose the **Program and Tool Path** property page and then select the **Linked Posts** tab.



- Select the **Link Other Posts to This Post** box and type **ZHEAD** in the **Head** text field.



Note that if there are any special G or M codes for needed for the beginning or end for your particular controller, add them here.

- Click **OK**.

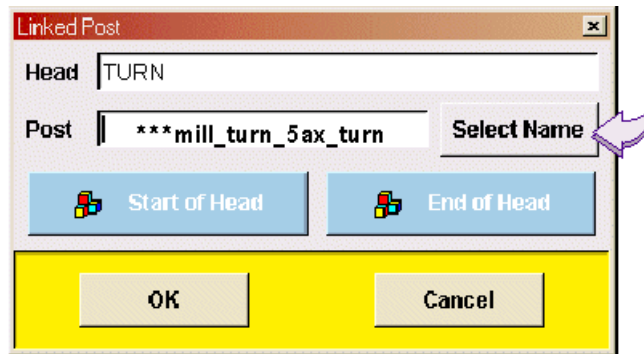
You will now specify the additional linked post for turning.



- ☐ Click **NEW**.



- ☐ Type **TURN** for the **Head** name and then choose the **Select Name** button and browse for the turning post which you created in **STEP 2** (**\*\*\*mill\_turn\_5ax\_turn.pui**).



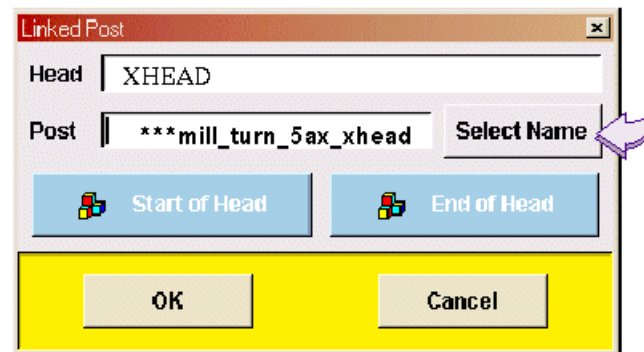
- ☐ Click **Open**.
- ☐ Click **OK**.

You will now specify the additional linked post for the **XHEAD**.

- ☐ Click **NEW**.



- ☐ Type **XHEAD** for the **Head** name and then choose the **Select Name** button and browse for the XHEAD post which you created in **STEP 3** (**\*\*\*mill\_turn\_5ax\_xhead.pui**).



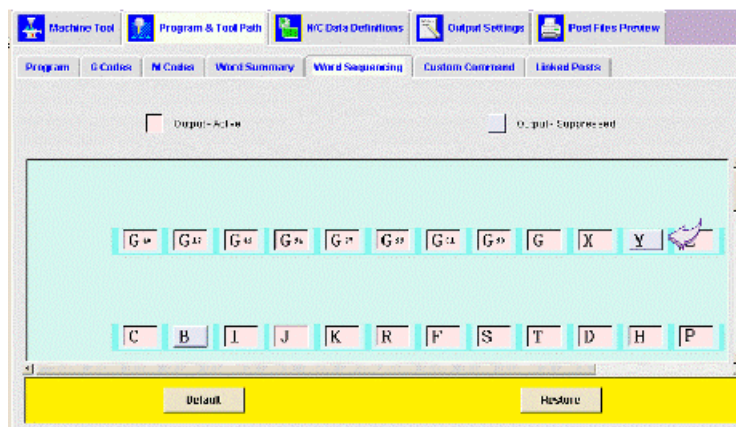
- ☐ Click **Open**.



- ☐ Click **OK**.

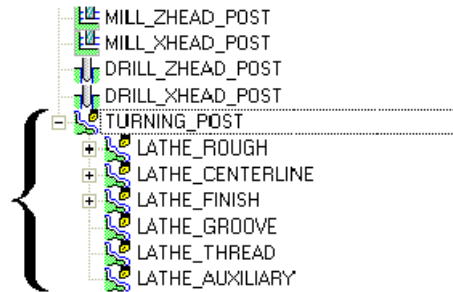
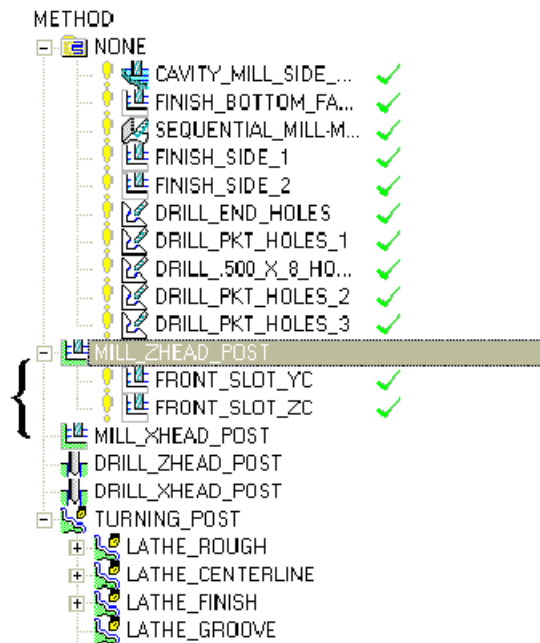
You will now add the parameters for the 4th axis.

- ☐ Click **Program and Tool Path** property page, then select the **Program** tab, the **Canned Cycle** component and then verify the fourth axis address in the **Common** parameters block.
- ☐ Click **OK**.
- ☐ If necessary, go to the **Word Sequencing** tab and suppress the **Y** and **J** addresses if not already suppressed.

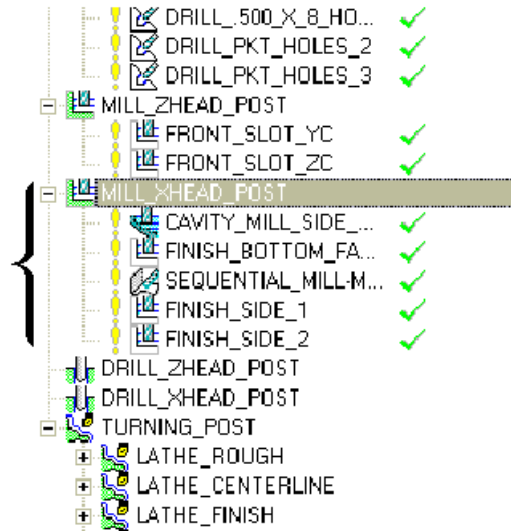


☐ DRILL\_XHEAD\_POST

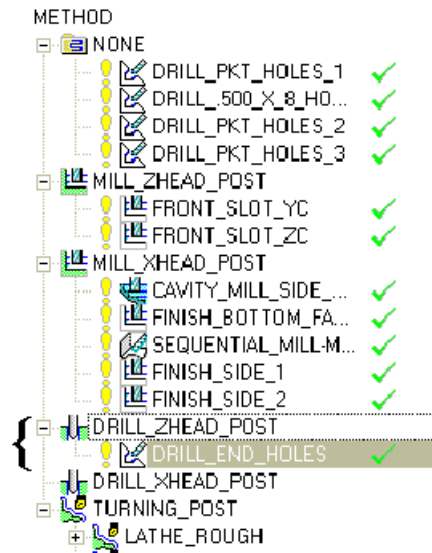
by choosing the Create Method icon and keying in the proper group object name.

☐ Move the existing turning tool paths to the **TURNING\_POST** method parent.

☐ Move all the **ZHEAD MILL** operations to **MILL\_ZHEAD** method parent.


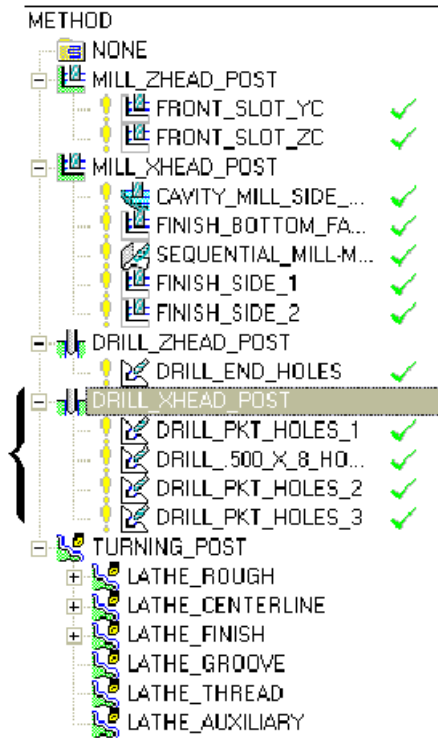
- ☐ Move all the **XHEAD MILL** operations to **MILL\_XHEAD** method parent.



- ☐ Move all the **ZHEAD DRILL** operations to **DRILL\_ZHEAD** method parent.



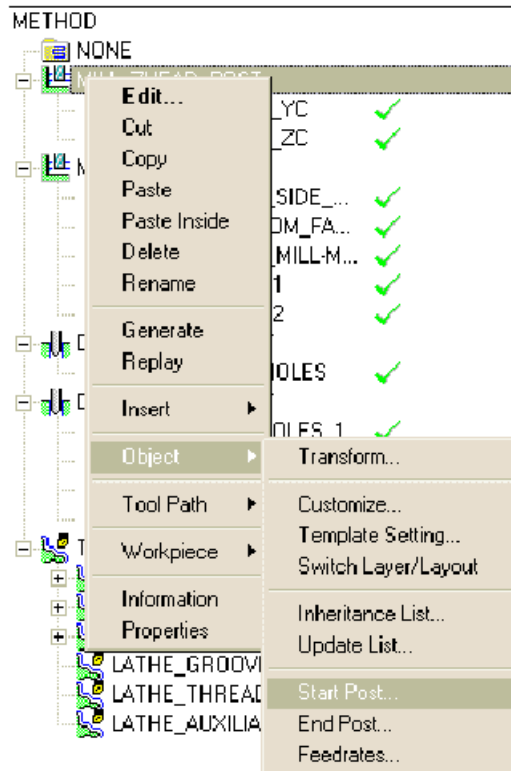
- Move all the **XHEAD DRILL** operations to **DRILL\_XHEAD** method parent.



## 6

**Step 6:** Assign the HEAD UDE to the Method parents.

- ☐ Choose the **MILL\_ZHEAD\_POST** method, right-click, choose **Object** and then **Start Events**.



The User Defined Events dialog is displayed.

- ☐ Select **Head** and then choose **Add**.

The Head dialog is displayed.

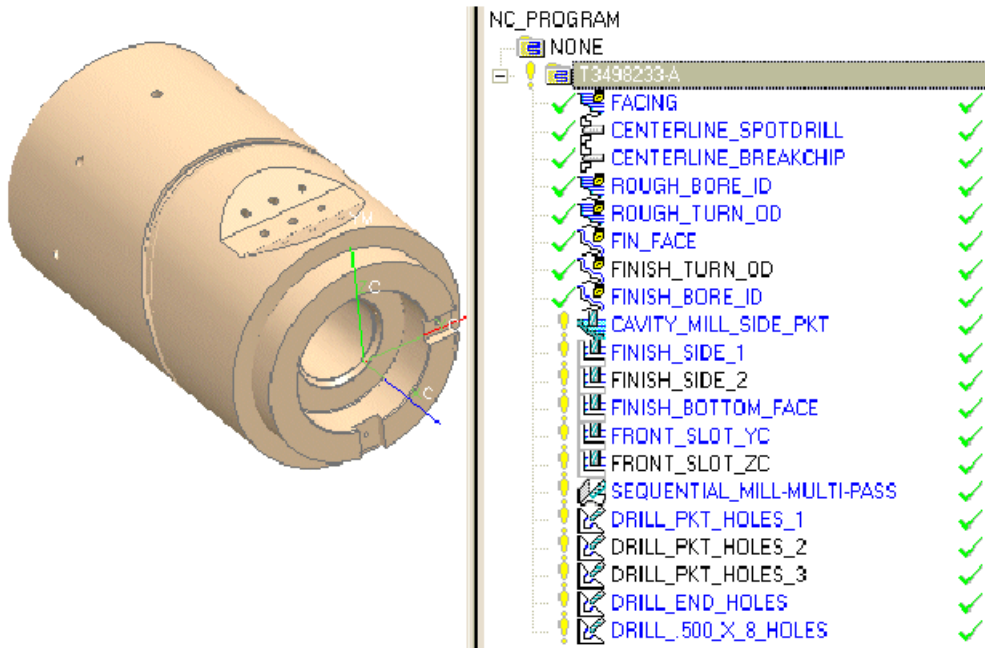
- ☐ Turn on Name Status and type in the name of the appropriate head, in this case **ZHEAD**.
- ☐ Choose **OK**.
- ☐ Repeat the process for **MILL\_XHEAD\_POST**. Assign **XHEAD** UDE for the **MILL\_XHEAD** method.
- ☐ Repeat the process for **DRILL\_ZHEAD\_POST**. Assign **ZHEAD** UDE for the **DRILL\_ZHEAD** method.
- ☐ Repeat the process for **DRILL\_XHEAD\_POST**. Assign **XHEAD** UDE for the **DRILL\_XHEAD** method.
- ☐ Repeat the process for **TURNING\_POST**. Assign **TURN** UDE to the **TURNING\_POST** method.

You have just created the all the post processors required for the mill-turn center and have assigned the UDE's to the proper heads. You also segregated your operations to coincide with the various methods which you have established. You will now test your post processor to verify the output.

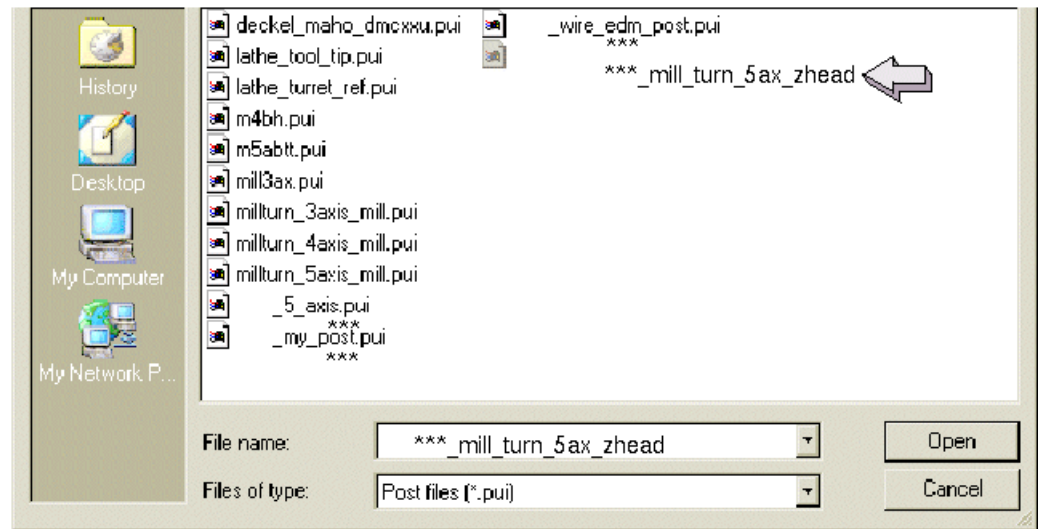
In order for your newly created post processor to be included in the NX Post processing window, you must modify the **template\_post.dat** file to include the new post processor. This file is located in the home post processor directory.

**Step 7:** Post process all tool paths.

- ☐ Change to the **Program Order** view of the Operation Navigator and expand all operations.



- Click T3498233 and then choose **Post Process** and browse for your post processor (**\*\*\*\_mill\_turn\_5ax\_zhead**).



- Choose the post processor and then **OK** on the Open Post processor dialog.
- Choose the **Apply** button at the bottom of the Post process dialog.

Your output will be similar to the following:

```
*
N0010 G40 G17 G90 G70
N0020 G92 X0.0 Z0.0
: 0030 T01 H01 M06
N0040 G97 S205 M03
N0050 G94 G00 G90 X9.0293 Z2.1033
N0060 X7.2 Z.8
N0070 G92 S2000
N0080 G96 S775 M03
N0090 G95 G01 X7.1 F.0135
N0100 X-.0469 F.009
N0110 X-.1469 F.04
N0120 G94 G00 Z.9
N0130 X7.2
N0140 Z.6
N0150 G95 G01 X7.1 F.0135
N0160 X-.0469 F.015
N0170 X-.1469 F.04
N0180 G94 G00 Z.7
N0190 X7.2
N0200 Z.4
N0210 G95 G01 X7.1 F.0135
N0220 X-.0469 F.015
N0230 X-.1469 F.04
N0240 G94 G00 Z.5
N0250 X7.2
N0260 Z.2
N0270 G95 G01 X7.1 F.0135
N0280 X-.0469 F.015
N0290 X-.1469 F.04

:1530 T13 M06
N1540 T03
N1550 G00 G90 X-17. Y3.3166 B0.0 C270. S1783 M03
N1560 G43 Z.5 H13
N1570 Z-9.8555
N1580 G01 X-15. Y0.0 Z-10. F30.
N1590 X-13.7356 Y-.0242 Z-10.6849 B-3.554 C269.759 F10.
N1600 X-13.7364 Y-.0219 Z-10.6845 B-3.438 C271.688 F9.6
N1610 X-13.7373 Y-.0201 Z-10.6816 B-3.309 C273.613
N1620 X-13.738 Y-.0189 Z-10.6799 B-3.201 C275.535
N1630 X-13.7385 Y-.018 Z-10.6831 B-3.152 C277.457
N1640 X-13.7385 Y-.0166 Z-10.6909 B-3.156 C279.38
N1650 Y-.0146 Z-10.7007 B-3.185 C281.305
N1660 X-13.7389 Y-.0131 Z-10.7047 B-3.161 C283.221
N1670 X-13.7396 Y-.0122 Z-10.6983 B-3.041 C285.132
N1680 X-13.7408 Y-.0114 Z-10.6834 B-2.848 C287.053
N1690 X-13.7422 Y-.01 Z-10.6621 B-2.616 C288.983
N1700 X-13.7436 Y-.0073 Z-10.6364 B-2.372 C290.924
N1710 X-13.745 Y-.0037 Z-10.6053 B-2.116 C292.866
N1720 X-13.7466 Y-.0027 Z-10.5601 B-1.772 C294.775
N1730 X-13.7482 Y-.0056 Z-10.505 B-1.382 C296.656
N1740 X-13.7497 Y-.0102 Z-10.4484 B-1.03 C298.531
N1750 X-13.7509 Y-.0131 Z-10.3971 B-.783 C300.422
N1760 X-13.7519 Y-.0134 Z-10.3504 B-.634 C302.323
N1770 X-13.7528 Y-.013 Z-10.303 B-.533 C304.216
N1780 X-13.7537 Z-10.2523 B-.458 C306.093
N1790 X-13.7545 Y-.0123 Z-10.1987 B-.435 C307.978
N1800 X-13.7554 Y-.0116 Z-10.1391 B-.407 C309.874
N1810 X-13.7562 Y-.011 Z-10.0741 B-.376 C311.769
N1820 X-13.757 Y-.0103 Z-10.0043 B-.343 C313.663
N1830 X-13.7577 Y-.0093 Z-9.9304 B-.312 C315.559
N1840 X-13.7584 Y-.0078 Z-9.8535 B-.289 C317.459
N1850 X-13.759 Y-.0056 Z-9.7745 B-.277 C319.365
N1860 X-13.7595 Y-.0031 Z-9.6934 B-.274 C321.274
N1870 X-13.76 Y-.0008 Z-9.6097 C323.181
N1880 X-13.7604 Y.0005 Z-9.5228 B-.27 C325.082
N1890 X-13.7608 Z-9.4323 B-.259 C326.974
```

This concludes the activity and the lesson.

## Summary

The ability to perform milling, drilling and turning operations, utilizing a mill-turn center, affords substantial productivity and cost savings in today's manufacturing environment. The ability to generate post processors for these types of machines, utilizing the Post Builder:

- Substantially reduces the time required to develop a post processor.
- Is easy to customize through the use of Custom Commands.
- Allows for multitudes of types of configurations



## Lesson

# 7 *Tcl basics for Post Builder*

### **Tcl Basics for Post Builder**

#### **Purpose**

This lesson describes the basics of the Tcl scripting language that is used by NX/Post and Post Builder.

#### **Objectives**

Upon completion of this lesson, you will be able to:

- Understand the basic structure and syntax of the Tcl scripting language.
- Write simple Tcl programs.

## Tcl

Tcl, stands for Tool Command Language, is an interpretive scripting language that is the NX language for user supplied rules and is used in NX/Post, Process Assistants, Shop Documentation, clsf file creation and the Post Builder.

Tcl has an additional component, Tk. Tk, stands for Tool kit, is considered to be an extension of Tcl and provides basic user graphic interface elements such as buttons, check boxes, and scroll bars.

Tcl was originally developed by John K. Ousterhout while at the University of California, Berkeley. Tcl is public domain software and is currently supported by the Scriptix Corporation.

Tcl can be activated by executing **ugwish** from the /mach/auxiliary directory.

## Tcl command structure

One of the basic building blocks of Tcl is the **command**. Commands instruct the Tcl interpreter to perform some type of task.

Commands are made up of words that include the name of the command and may also include options and or arguments.

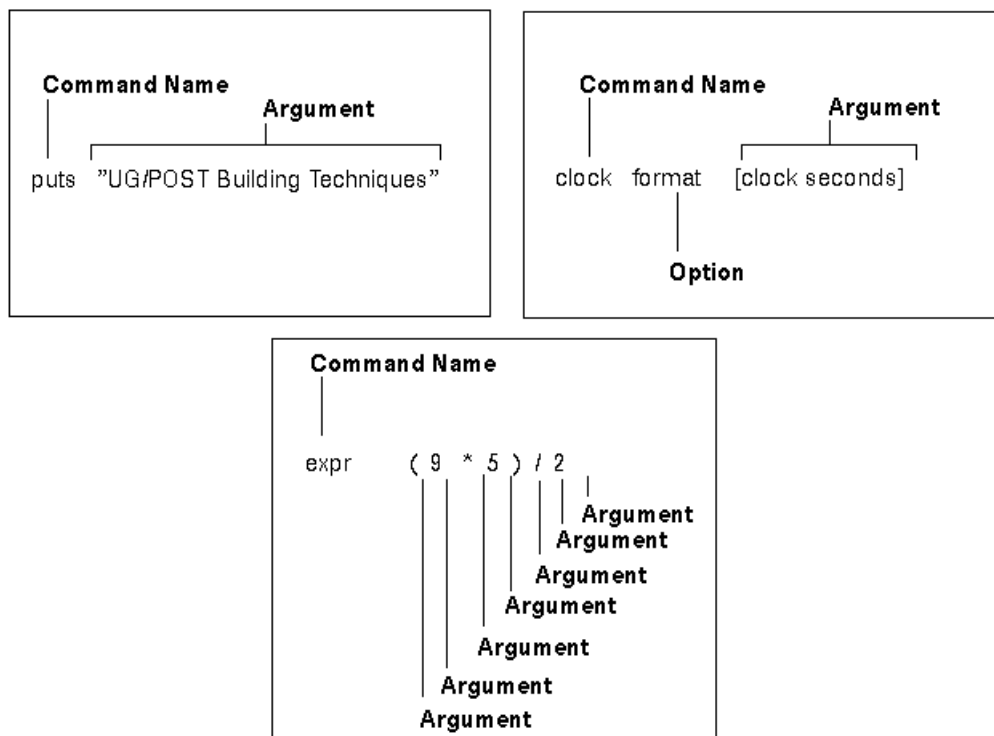
- Options give the interpreter detailed instructions of the task that the command must perform
- Arguments are any type of information that can be processed, changed, or used in some way by the command

In summary, a **command** is an instruction consisting of three parts: name, option, and argument.

A typical command syntax is:

command name option argument 1 argument 2 argument n

### Examples of commands with options and argument components



## Tcl scripts

A Tcl **script** consists of a sequence of commands separated by either a "new line" or a semicolon.

Comments are preceded by the "#". The following example illustrates the use of a comment and the concept of a script:

```
Script { #If the value of X is less than 3, then print } Comment
        #the following message: "X is too small"
        if { $X < 3 } {
            puts stdout "X is too small"
            set X 3
        }
```

## Tcl control of word structure

Tcl words break at "white space(s)", tabs and semi-colons, except in the following situations:

- double quotes prevent breaks and must be used in strings that have spaces:

```
set X "this is a word which has spaces"
```

- Curly braces prevent breaks and substitutions:

```
set Y {nested {}braces}
```

- Backslashes quote special characters:

```
set Z word/ with/ /]$/ and/ space
```

- Substitutions do not change word structure:

```
set abc "two words"
```

```
set b $abc
```

```
b is now equal to the string "two words"
```

## Tcl variables

A Tcl variable can be thought of as a named place holder of information. A variable can store either numbers or strings. A number can be thought of as an item that can be equated or has value. A string can be considered to be a sequence of characters consisting of alphabetic letters, numbers or symbols, or any combination of the three.

A Tcl variable name must start with an alphabetic letter and can consist of alphanumeric characters. Spaces are not allowed in the variable name. The variable name is case sensitive as well.

All variable values are stored as strings. Special functions exist to perform calculations on numerical represented strings.

Tcl allows for local and global variables. Local variables are used inside of a **proc**(also called a procedure which is like a subroutine and will be discussed later). When the proc is called, the variable is created. When the proc is completed, the variable is removed. A **global** variable is just the opposite of the local variable. It allows the storage of information between calls to a proc. The global variable has the same name regardless of location in the program, whether in the main part of the program or within various procs.

## Tcl mathematical expressions

Tcl mathematical expressions are performed by the `expr` command. Basic operators used in expressions are:

- `+` addition
- `-` subtraction
- `/` division
- `*` multiplication

More complex operators, such as trigonometric and logarithmic functions are also available.

## Variable examples

Variable name

X

x

this\_is\_a\_Variable

pre\_drill

PRE\_DRILL



## Tcl variable definitions

In most computer languages such as FORTRAN or "C", variables are set by the use of the "=" sign, i.e. XYZ=49 or XX=5\*5. With Tcl, variables are defined by using the set command.

Tcl Variable definition example:

```
set XYZ 49
set XX [ expr 5*5 ]
```

## Variable substitution

Variable substitution will cause a command to use the value of the variable instead of the variable name in the command. For example, if `A=2` and you want to multiply `A` by 10 you would write the expression as `expr A*10`. This would be interpreted by Tcl as multiply the string `A` by 10 rather than multiple the contents of `A` by 10. In order for Tcl to use the value of the variable instead of the name, the `"$"` character must precede the variable's name. The correct way to write the expression would be `expr $A*10`.

## Variable substitution examples

Sample Command	Result
set a Uni	a=Uni
set b graphics	b=graphics
set c \$a\$b	c=\$a\$b=Unigraphics
set b 32	b=32
set a b	a=b
set a \$b	a=32
set a \$b+\$a+\$b	a=\$b+\$a+\$b=32+32+32
set a \$b.3	a=\$b.3=32.3
set a \$b9	invalid since b9 is not a defined variable

## Activity — Tcl basics

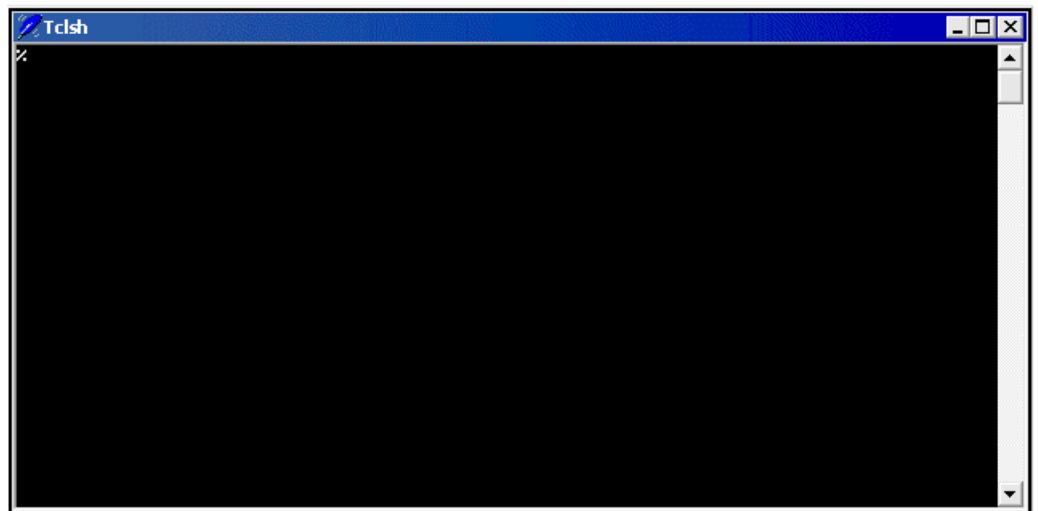
In this activity you will start the Tcl Shell creating a window that will let you create and test Tcl commands.

**Step 1:** Start the Tcl Shell.

- ☐ Choose **Start**→**Programs**→**Tcl**→**Tclsh**.



The Tcl Shell window is displayed.



The Tcl Shell may be used for executing simple one line commands or multiple line programs, stored as a text file. Notice the "%" as the prompt.

**Step 2:** Create the command(s) to evaluate the following:

$$4 + (5*6)/2 + (7*28)/4$$

- ☐ In the Tcl Shell window type the following:

```
expr (4 + (5*6)/2 + (7*28)/4)
```

followed by a carriage return. The result is 68.

**Step 3:** Set the following variables:

```
mom_sys_zaxis_max_limit = 9999.9999
```

- In the Tcl Shell window type the following: set mom\_sys\_zaxis\_max\_limit 9999.9999 followed by a carriage return.

**PI = 3.1415926536**

- In the Tcl Shell window type the following:

```
set PI 3.1415926536
```

followed by a carriage return.

**DEGRAD = PI/180**

- In the Tcl Shell window type the following

```
set DEGRAD [expr $PI/180]
```

hint: remember that you want to evaluate the contents of PI, hence the "\$" character must precede the variable's name.

The result is 0.01745329252.

This concludes this activity.

## Tcl procedures and functions

Procedures are the same basic routines in different contexts that can be used throughout a program. Procedures are analogous to subroutines and make programs easier to create and modify.

The structure of a procedure follows:

**proc procedure name arguments body**

the proc command defines the procedure, for example in the command, proc sub1 x {expr \$b-1}, sub1 is the procedure

arguments within procedures may have defaults

arguments may also contain local as well as global variables

As previously stated, Tcl allows for local and global variable usage. Local variables are used inside of a procedure. When the procedure is called, the variable is created. When the procedure is complete, the variable is removed. A **global** variable is the opposite of the local variable. It allows the storage of information between calls to a procedure. The global variable has the same name regardless of location in the program, whether in the main part of the program or whether contained within various procedures.

Hint: When naming procedures, it is suggested that you name the procedure with capital letters. This will make the procedure easier to locate when writing scripts.

Example of a procedure:

Example of a procedure:

```
#####
proc PB_CMD_ptp_size { } {
#####
# Output the current size of the ptp output file.
# Add this to the end of the End of Program event.

global ptp_file_name
global mom_sys_control_out
global mom_sys_control_in

set ci $mom_sys_control_in
set co $mom_sys_control_out

# Check the file size
MOM_close_output_file $ptp_file_name
set ptp_size [file size $ptp_file_name]
MOM_open_output_file $ptp_file_name

# Put a message for the file size in the ptp file
set ptp_feet [expr $ptp_size/120.]
MOM_output_literal "$co PTP file size = $ptp_size bytes/
[format "%5.1f" $ptp_feet] feet $ci"

}
```

Functions have the same functionality as a procedure, but they create a result and return a value. A return command must be used prior to the last command in the program.

## Tcl I/O

Output of a string to the runtime window of Tcl is achieved by using the `puts` command. For example, `puts stdout " Post Processor"` outputs the following on your screen: " **Post Processor**". To output the value of a variable, use the "\$" character prior to the variable name. An example would be the following: `puts stdout $name`.

To input a string value from the runtime window of Tcl, use the **gets** command. For example, `gets stdin var` will read the string value from the variable "var".



## Tcl special characters

Use the `'` preceding the character for the output of special characters such as `$`, `/`, `tab`, `LF`.

```
/ puts stdout "/"
```

```
$ puts stdout "$"
```

```
tab puts stdout "/t"
```

```
LF puts stdout "/013"
```

## Construct a simple procedure

In this activity, you will create and test a simple procedure that will calculate the circumference of a circle. You will then output the results in the Tcl Shell window. You will use the Notepad editor to create your procedure as a text file and will then "source" the procedure into the Tcl Shell window.

**Step 1:** Use the Notepad editor to create a text file.

- ☐ Open a Notepad editor window.

**Step 2:** Create a Tcl procedure that will calculate the circumference of a circle and output the results.

- ☐ Create the start of the procedure. You will call the procedure CIRCUMFERENCE. Type the following at the beginning of text file:

```
#=====
proc CIRCUMFERENCE { } {
#=====
#
#Output the circumference based on the input diameter
#
```

- ☐ Create the global variable diameter. This variable is global so that it may be used outside of the procedure. Type the following after the last line from the previous step action item:

```
global diameter
```

- ☐ Create a variable, PI, for the value of pi; then create a variable circ, that represents the circumference. Type the following after the last line from the previous step action item:

```
set PI 3.1415926536
set circ [expr $PI * $diameter]
```

Note that the "\$" precedes the variable PI and **diameter** since you want to use the value of those two variables.

- ☐ Output the circumference value. Type the following after the last line from the previous step action item:

```
puts "Circumference is $circ"
```

- End the procedure, create an inquiry for the diameter of the circle and execute the procedure. Type the following after the last line from the previous step action item:

```
#end the procedure
}
#inquire for diameter value
puts stdout " Enter Diameter Value"
#Get input from the keyboard
gets stdin diameter
#execute the procedure
CIRCUMFERENCE
```

**Step 3:** Save the file and execute the procedure from the Tcl Shell window.

Your procedure should be similar to the following:

```
=====
proc CIRCUMFERENCE { } {
#=====
# Output the circumference based on input diameter.

global diameter

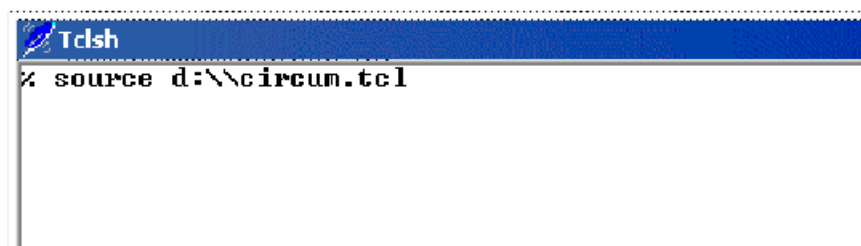
set PI 3.1415926536
set circ [expr $PI * $diameter]
puts "Circumference is $circ"
#End the procedure
}

puts stdout " Enter Diameter Value"
gets stdin diameter
CIRCUMFERENCE
```

- From the Notepad tool bar, File -> Save the file in your home directory with the name circum.tcl.
- Execute the procedure from the Tcl Shell by typing the following at the prompt

```
source "pathname filename"
```

where pathname is the directory specification (use "/" for every "/" occurrence) and the filename is circum.tcl



- At the prompt for "Enter Diameter Value", key in 15

The value returned is 47.123889804

- ☐ – Now edit the `circum.tcl` file to shorten the answer to show only three places to the right of the decimal and edit the **puts** command to output "The circumference is...."

This completes this activity.

## Tcl flow control structures

Flow control structures (also referred to as conditionals and looping) are special Tcl commands that allow control of the program execution. Using conditional structures, the program decides which commands are executed and which commands are bypassed. With looping structures, commands can be repeated for repetitive tasks or functions. Control structures test a statement to see if it is true or false. If the statement is true then a designated instruction or group of instructions will be executed. If false, an alternate designated instruction or group of instructions will be executed or bypassed. You can generally use control structures to handle just about any set of conditions that you may encounter in writing postprocessors.

The **conditional** "if" structure uses the following basic equivalence operators:

- == equal
- != not equal to
- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to

Example of "if" construct:

```
if {$mom_sys_cir_vector == "Vector - Arc Center to Start"} {
set mom_prev_pos($cir_index) 0.0
set mom_pos_arc_center($cir_index) $pitch
} elseif {$mom_sys_cir_vector == "Vector - Arc Start to Center"}{
set mom_prev_pos($cir_index) $pitch
set mom_pos_arc_center($cir_index) 0.0
} elseif {$mom_sys_cir_vector == "Unsigned Vector - Arc Center / to Start"} {
set mom_prev_pos($cir_index) 0.0
set mom_pos_arc_center($cir_index) $pitch
} elseif {$mom_sys_cir_vector == "Absolute Arc Center"} {
set mom_pos_arc_center($cir_index) $pitch
}
}
```

- command name
- initialization statement which initializes the counter that keeps track of the number of loops
- test statement decides whether to execute the body of the loop
- re-initialization statement keeps track of the counter

- body of loop contains commands that are executed when the test statement is true

Example of "for" construct:

```
#####
proc VMOV { n p1 p2 } {
#####
#
# n, p1 and p2 are variables that are passed through to the proc
# when the procedure is called
#
upvar $p1 v1 ; upvar $p2 v2
#
#"upvar" substitutes the value of p1 to the v1 variable
#
for {set i 0} {$i < $n} {incr i} {
set v2($i) $v1($i)
}
}
```

- command name
- test statement
- the body of loop

Example of "while" construct:

```
#####
proc ROTSET { angle prev_angle dir kin_leader sys_leader } {
#####

#This procedure will take an input angle and format for a specific #machine.
#angle = angle to be output.
#prev_angle = previous angle out. It should be mom_out_angle_pos
#dir can be either MAGNITUDE_DETERMINES_DIRECTION #or SIGN_DETERMINES_DIRECTION
#kin_leader = leader (usually A, B or C) defined by postbuilder
#sys_leader = leader that is created by rotset. It could be C-.
#
upvar $sys_leader lead

while {$angle < 0.0} {set angle [expr $angle+360.0]}
while {$angle >= 360.0} {set angle [expr $angle-360.0]}
if {$dir == "MAGNITUDE_DETERMINES_DIRECTION"} {
while {[expr abs($angle-$prev_angle)] > 180.0} {
if {[expr $angle-$prev_angle] < -180.0} {
set angle [expr $angle+360.0]
} elseif {[expr $angle-$prev_angle] > 180.0} {
set angle [expr $angle-360.0]
}
}
} elseif {$dir == "SIGN_DETERMINES_DIRECTION"} {
set del [expr $angle-$prev_angle]
if (($del < 0.0 && $del > -180.0) || $del > 180.0) {
set lead "$kin_leader-"
} else {
set lead $kin_leader
}
}
}
```

```
    return $angle
}
```

The "switch" command consists of five parts:

- command name
- command options
- a string to be matched
- a collection of strings that is compared with the first string
- a set of instructions for each compared string; the instructions being executed if the string is matched
- Example of "switch" construct:

```
switch $mom_rotation_mode {
NONE {
set angle $mom_rotation_angle
set mode 0
}
ATANGLE {
set angle $mom_rotation_angle
set mode 0
}
ABSOLUTE {
set angle $mom_rotation_angle
set mode 1
}
INCREMENTAL {
set angle [expr $mom_pos($axis) + $mom_rotation_angle]
set mode 0
}
}
```

## Activity — Tcl flow constructs

In this activity you will write a script that consists of a main program and two (procedures) subprograms. The main program asks the user to calculate the area of either a rectangle or a circle. The first procedure pertains to the rectangle and requires the input of the length and width and then calculates the area. The second procedure pertains to the circle, requires the radius as input, defines Pi and then calculates the area of the circle.

**Step 1:** Step 1 Use the Notepad or Wordpad editor to create a text file.

- ☐ – Open a Notepad or Wordpad editor window.

**Step 2:** Step 2 Create the procedure, named RESULTS which will output the results.

- ☐ Create the procedure. You will call the procedure **RESULTS**.

```
#=====
proc RESULTS { } {
#=====
#
global area
#
if {$area > "0" } {
puts stdout " Area is [format "%2.2f" $area]"
} else {
puts stdout "Invalid Entry!"
}
}
```

**Step 3:** Create the procedure for calculating the area of the circle

Create the procedure for calculating the area of a circle. You will call the procedure CIRCLE. Begin typing the procedure with a new line immediately after the "]" from the previous step action item.

```
proc CIRCLE { } {
#=====
#
#Calculate the area of a circle
#
global area
#
#set the variable for PI
#
set PI [expr 2.0 * asin(1.0)]
#
#input the radius value
#
set radius 0
puts stdout " Enter the radius value"
gets stdin radius
set area [expr $PI * $radius * $radius]
RESULTS
}
```



**Step 4:** Create the procedure to calculate the area of a rectangle

- ☐ Create the procedure for calculating the area of a rectangle. You will call the procedure RECTANGLE. Begin typing the procedure with a new line immediately after the "}" from the previous step action item.

```
#=====
proc RECTANGLE { } {
#=====
#
#Calculate the area of a rectangle
#
global area
#
#set variables for length and width; input values for
#the variables
#
set length 0
set width 0
puts stdout " Enter length "
gets stdin length
puts stdout " Enter width "
gets stdin width
set area [expr $length * $width]
RESULTS
}
```

**Step 5:** Create the Main program.

- ☐ Create the Main program, that asks the type of area to calculate.

```
set in ""
puts stdout " Select c for circular area, or r for /
rectangular area"
gets stdin in
if {$in == "Q" || $in == "q" || $in == "quit"} exit
switch $in {
r { RECTANGLE }
c { CIRCLE }
R { RECTANGLE }
C { CIRCLE }
default { puts stdout " Invalid Entry!"}
}
```

**Step 6:** Step 6 Save the file and execute the procedure from the Tcl Shell window.

- ☐ From the Notepad tool bar, File -> Save the file in your home directory with the name results.tcl.

- ☐ Execute the procedure from the Tcl Shell window by typing the following at the prompt:

```
source "pathname filename"
```

where pathname is the directory specification (use "/" for every "/" occurrence) and the filename is results.tcl

- ☐ At the prompt "Select c for circular area, or r for rectangular area", key in **c**, followed by a carriage return
- ☐ At the prompt " **Enter the radius value**", key in 15.5, followed by a carriage return.

The value returned is 754.767635025.

This completes this activity.

## Tcl formats

Tcl provides numerous commands for processing lists. A list is a way of categorizing items or objects and then giving them a single name. Commands are available to add objects to a list, access elements in the list, access information concerning lists, to search for items in an list, and finally to sort lists.

To create a list, use the list command. To add elements to a list use the lappend command. To access elements in a list, use the lindex command. To replace elements in a list use the lreplace command. To insert elements in a list use the linsert command.

## Tcl and Unigraphics

Extensions of Tcl were developed for specific applications such as NX/POST, Shop Documentation, Process Assistants, Libraries and User Defined Features.

Tcl can call UFUNC routines and UFUNC routines can call GRIP routines. Tcl is easy to use and is extensible using the "C" language. Tcl currently can be accessed through the CAM modules such as NX/POST or Shop Documentation.

The Custom Command feature of the Post Builder allows the insertion of your own custom Tcl procedures into the postprocessor or procedures that are provided in the Custom Command library. The Post Builder provides the procedure name and correct syntax prior to and after the body of the procedure.

Common procedures are provided, and new procedures are added to each release of the Post Builder, however, you may be required to construct your own procedure to output specific formats required for special applications.

## Tcl reference manuals

**Practical Programming in Tcl and Tk** by Brent B. Welch ISBN 0-13-616830-3  
SAMS Teach Yourself Tcl/Tk by Venkat V.S.S. Sastry & Lakshmi Sastry  
ISBN 0-672-31749-4 Tcl/Tk Programmer's Reference by Christopher Nelson  
ISBN 0-07-212004-5

## Summary

Summary: Tcl is a scripting language that is extensively used in Post and Post Builder. The Custom Command feature of the Post Builder allows the insertion of custom Tcl procedures that can be created for applications such as custom output formats or specific custom machine tool/controller functionality. The following Tcl functionality is used in the creation of Custom Commands:

- Variables
- Mathematical expressions
- Procedures
- Input/Output statements
- Flow control structures

## Lesson

# 8 *Customize a post processor with Post Builder*

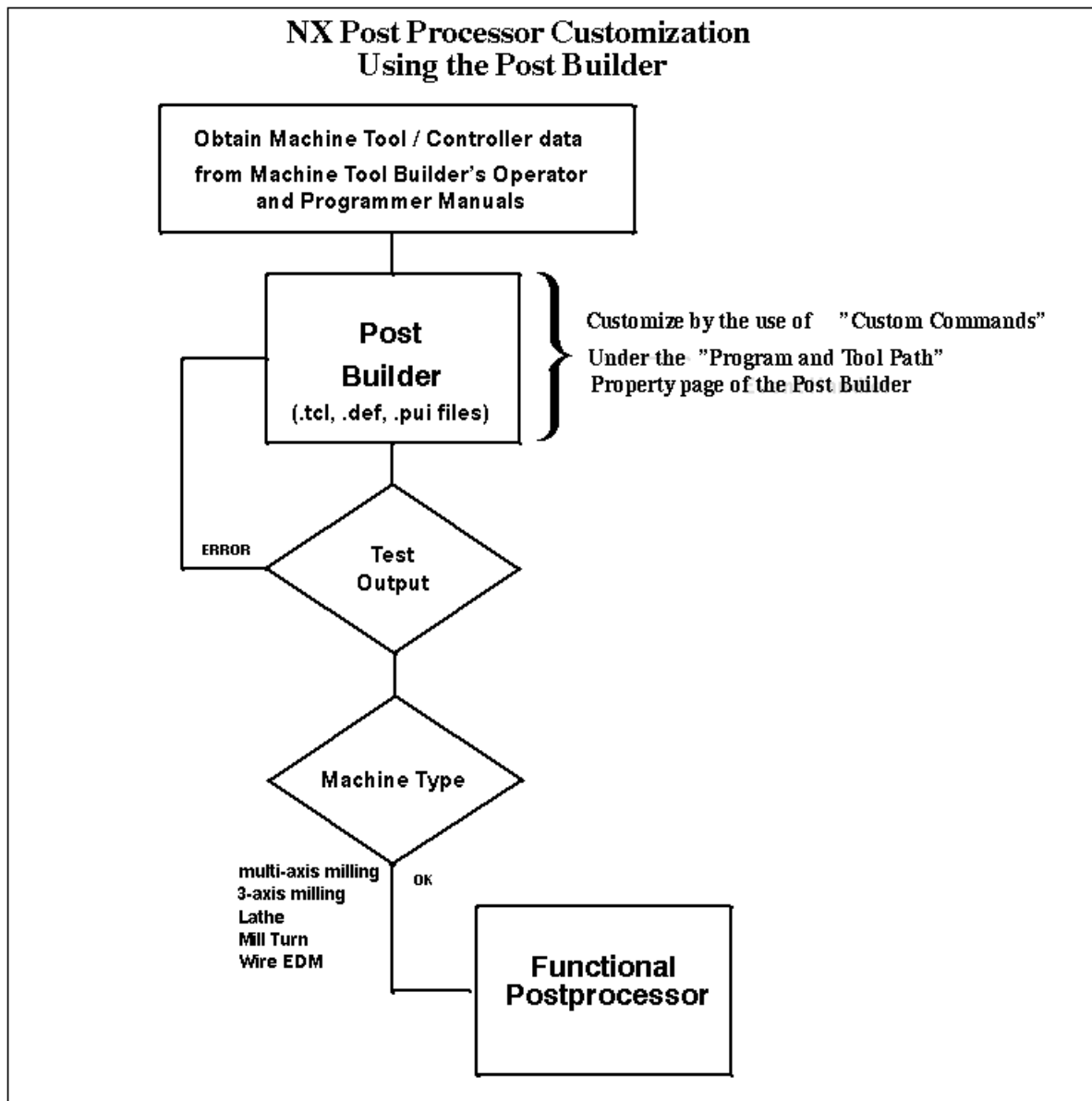
### Purpose

This lesson describes the procedures of customizing post processors through the use of the **Post Builder**.

### Objective

Upon completion of this lesson, you will be able to:

- Use the Post Builder to customize post processors through the use of the Custom Command feature.

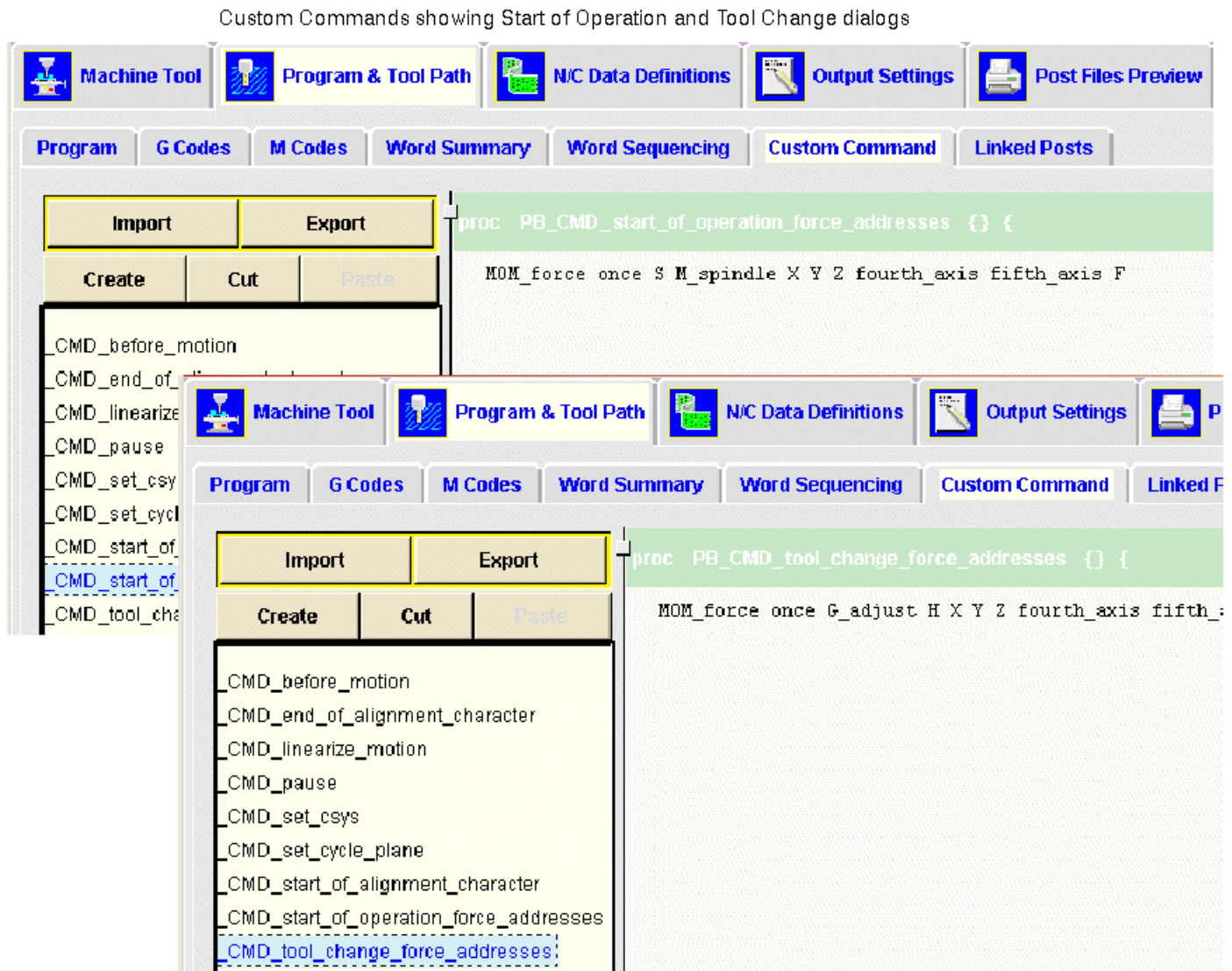


## Customize a post processor with the Post Builder

In the majority of cases, **Post Builder** allows you to generate the data which is necessary for your machine tool/controller. However, you may find that in some situations, such as generating a custom header at the beginning of a program, the Post Builder will not be able to give you the specific type of output that you may need.

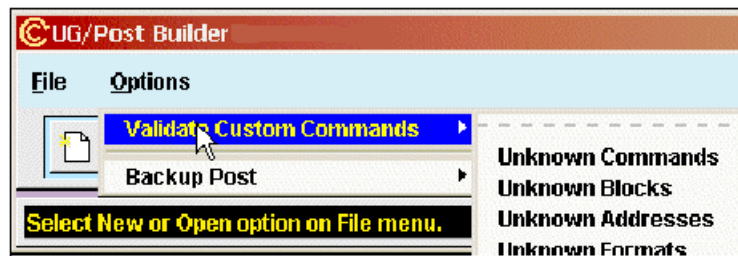
When this type of situation is encountered, the **Custom Commands** feature found in the **Program and Tool Path** property page will allow you to generate the output that is required.



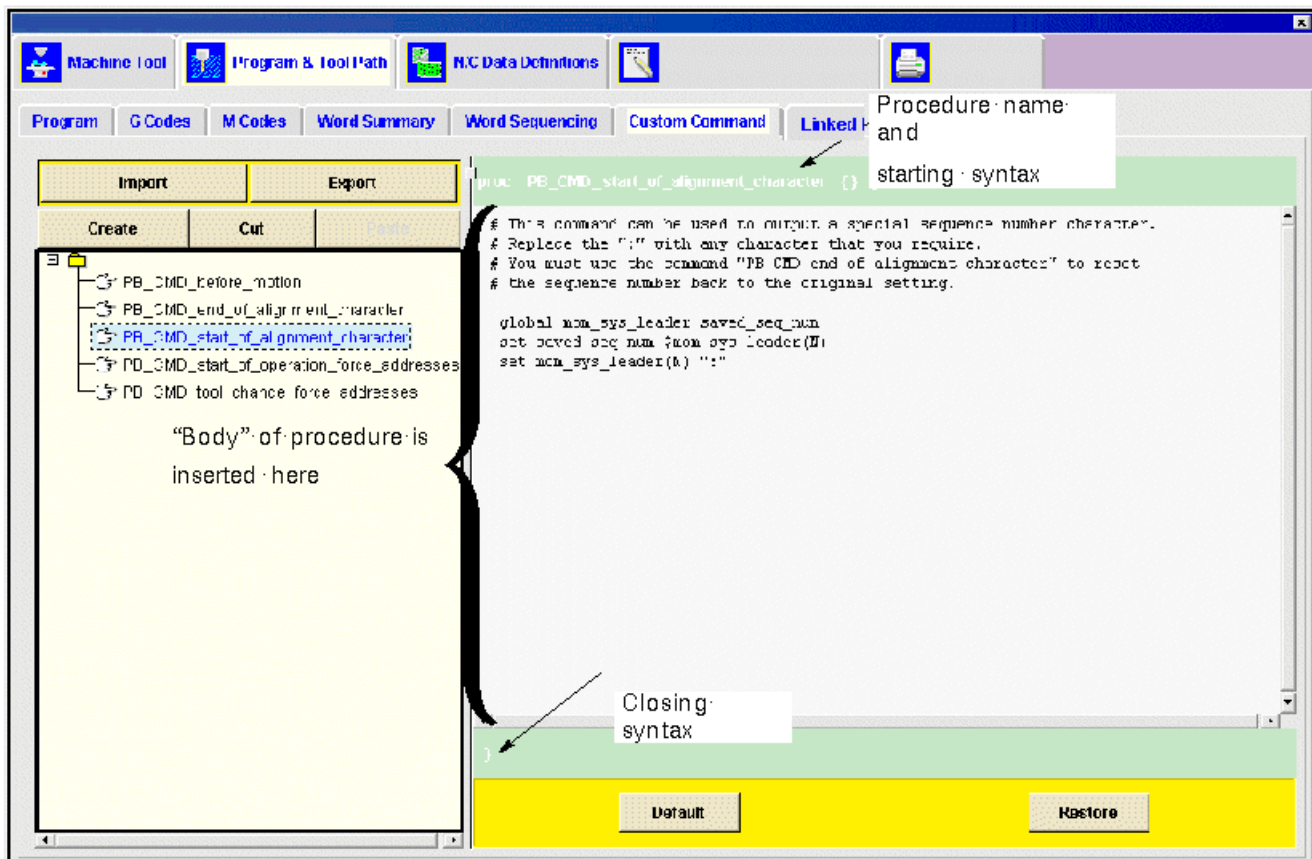


The **Custom Command** feature allows the insertion of your own custom Tcl procedures into the post processor or procedures that are provided in the Custom Command library. The **Post Builder** provides the **procedure name** and correct syntax prior to and after the body of the procedure. Post Builder checks the syntax of the user created custom command and displays syntax errors as found.

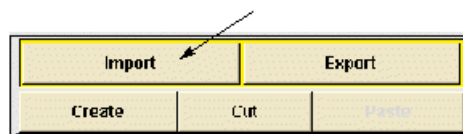
Note that the **Post Builder** checks the correctness of the syntax of the proc's body automatically. You can, as an option check for unknown commands, unknown blocks, unknown addresses and unknown formats for validation through selecting, **Options® Validate Custom Commands**.



## The Custom Command Property Page



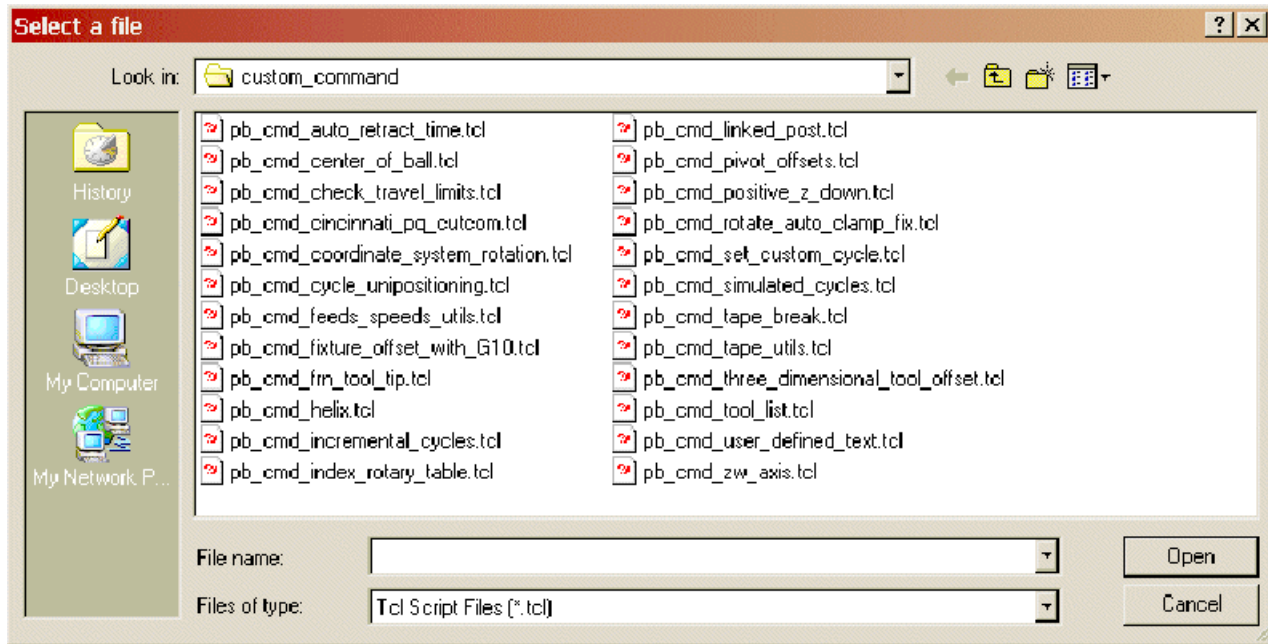
The **Import** button allows the importing of useful existing procedures that may be added to the various Event markers. These existing procedures normally have instructions including how the procedure functions.



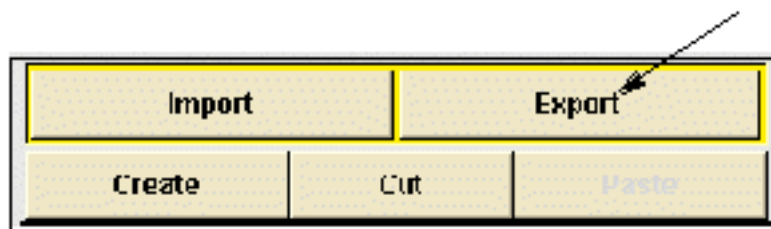
When you select the **Import** button, the **Select a File** dialog is displayed. The default directory is **custom\_command**, which contains numerous existing procedures. You may also browse to any directory to import any of your own exported files containing Custom Commands. Any file that is imported is

checked for Tcl errors based on the options that have been previously set for syntax checking.

Files located in the custom command directory begin with **pb\_cmd** and have a **.tcl** extension.



The Export button allows you to export custom commands that have been created or modified to a specific file. This is a convenient method of saving frequently used procedures that you create as you are building your post processor.



- Import a Custom Command that outputs the current size of the machine code output file at the end of the file
- Cut and paste text from a text file, that creates a Custom Command to create blocks of header information at the beginning of your posted output file
- Position the Custom Commands which you created to output the information requested at the beginning and end of the output file

- Export the Custom Command that creates a block of header information as a file into the Custom Command directory



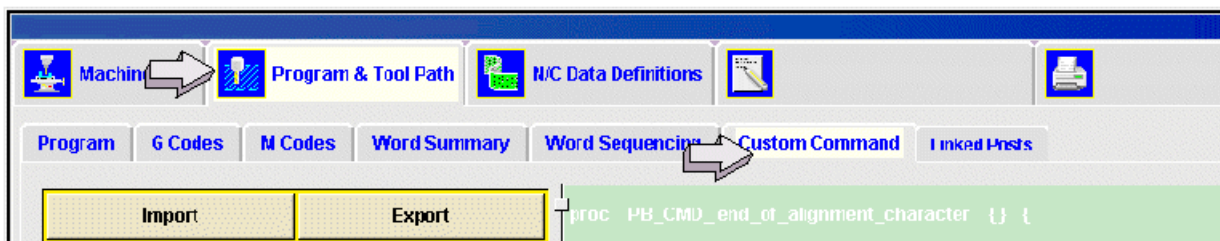
## Activity — Custom commands

In this activity you will:

- Import a Custom Command that outputs the current size of the machine code output file at the end of the file
- Cut and paste text from a text file, that creates a Custom Command to create blocks of header information at the beginning of your posted output file.
- Position the Custom Commands which you created to output the information requested at the beginning and end of the output file
- Export the Custom Command that creates a block of header information as a file into the Custom Command directory You will be using the post processor, `***_my_post` that you created in a previous activity

**Step 1:** Enter the Program and Tool Path and then Custom Command section of Post Builder.

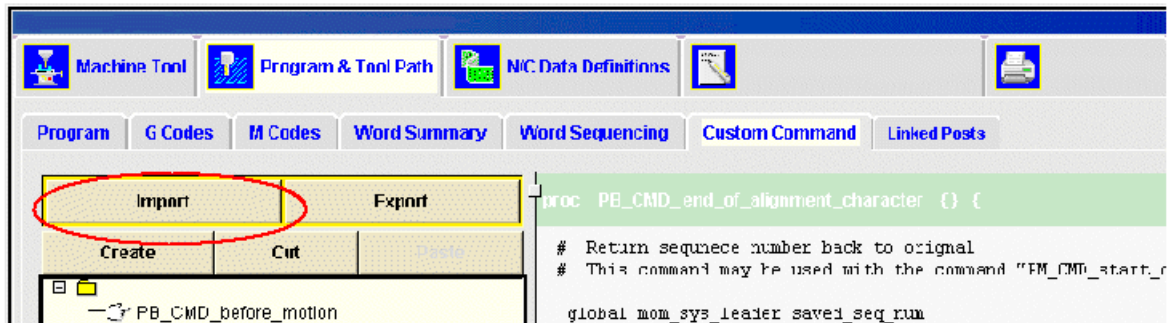
- ☐ If necessary, open `***_my_post`, where `***` represents your initials.
- ☐ Select the **Program and Tool Path** property page from the Post Builder dialog.
- ☐ Select the **Custom Command** property page.



**Step 2:** Import a Custom Command that outputs the current size of the machine code output file.

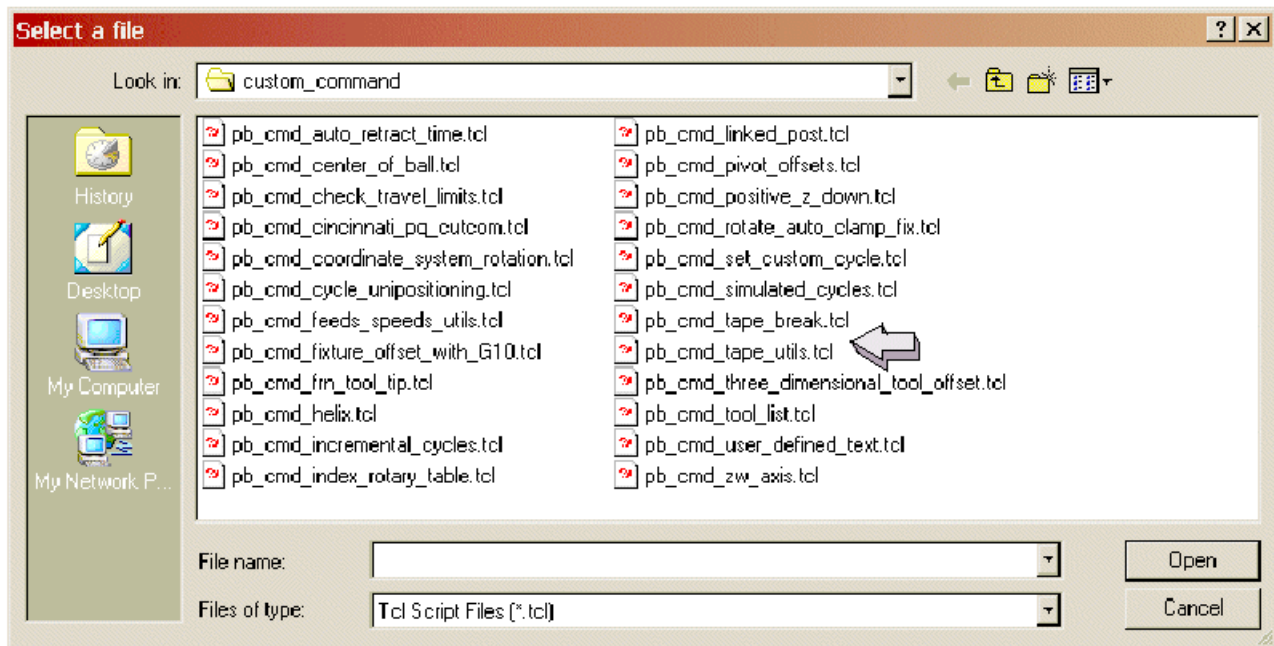


- ☐ Click **Import** from the **Custom Command** page.



The **Select a file** dialog is displayed.

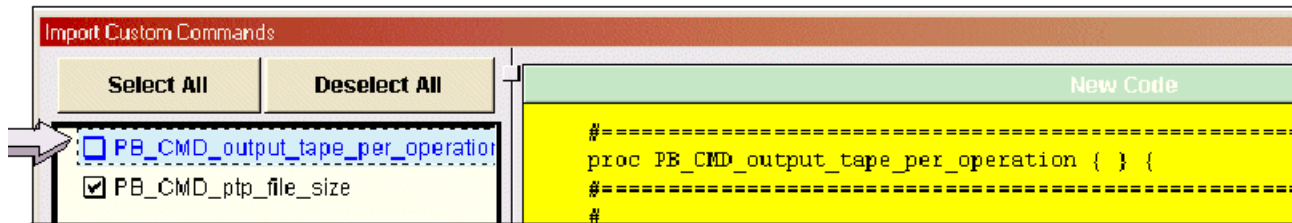
- ☐ Choose **pb\_cmd\_tape\_utils.tcl**.



- ☐ Click **Open**.

Note that this file contains the procedures, **PB\_CMD\_output\_tape\_per\_operation** and **PB\_CMD\_ptp\_file\_size**. Both procedures have a check marked box, which implies that they will both be placed into your post. Since you only want to use the **PB\_CMD\_ptp\_file\_size** procedure, you will need to uncheck the procedure **PB\_CMD\_output\_tape\_per\_operation**.

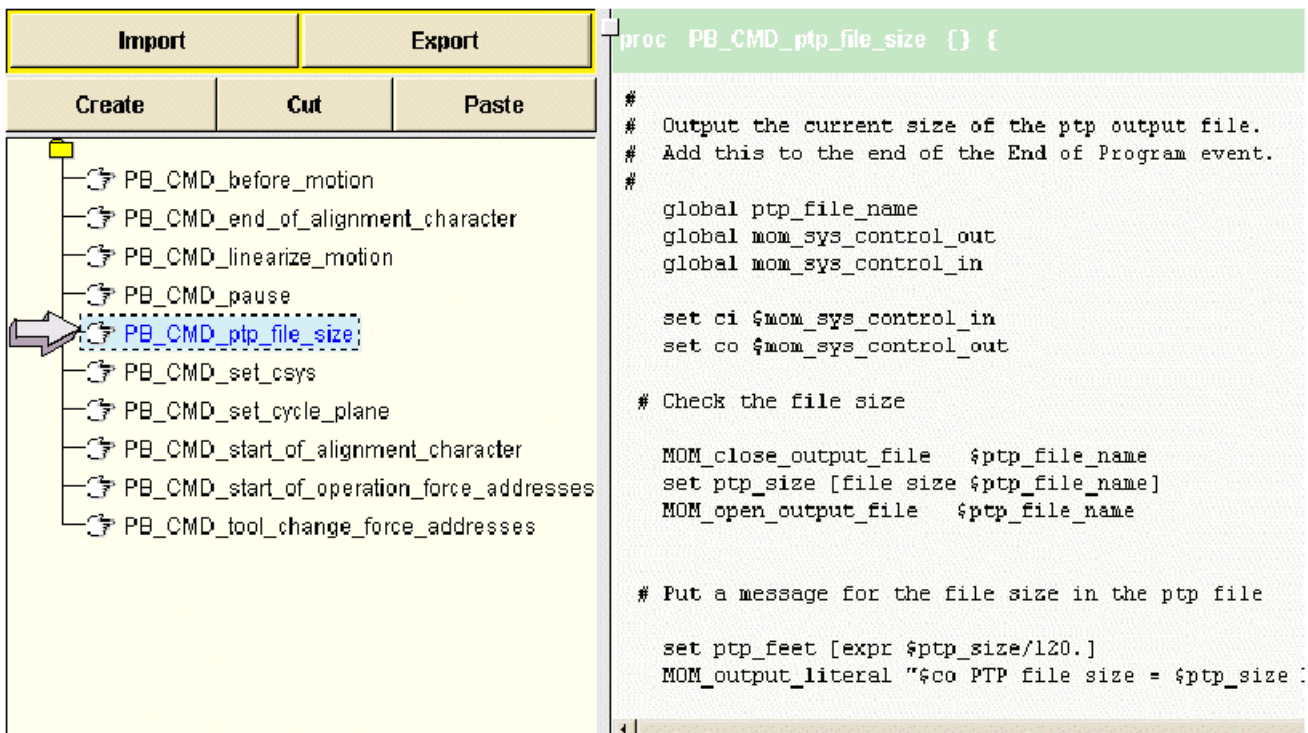
- Double-click on the procedure **PB\_CMD\_output\_tape\_per\_operation** to deselect.



The procedure, **PB\_CMD\_output\_tape\_per\_operation** is deselected.

- Click **OK**.

The procedure name is displayed, highlighted in the Component Window of the Custom Command property page. The contents are displayed in the Parameter Window.



**Step 3:** Create a Custom Command which will create blocks of header information at the beginning of your program using cut and paste procedures.

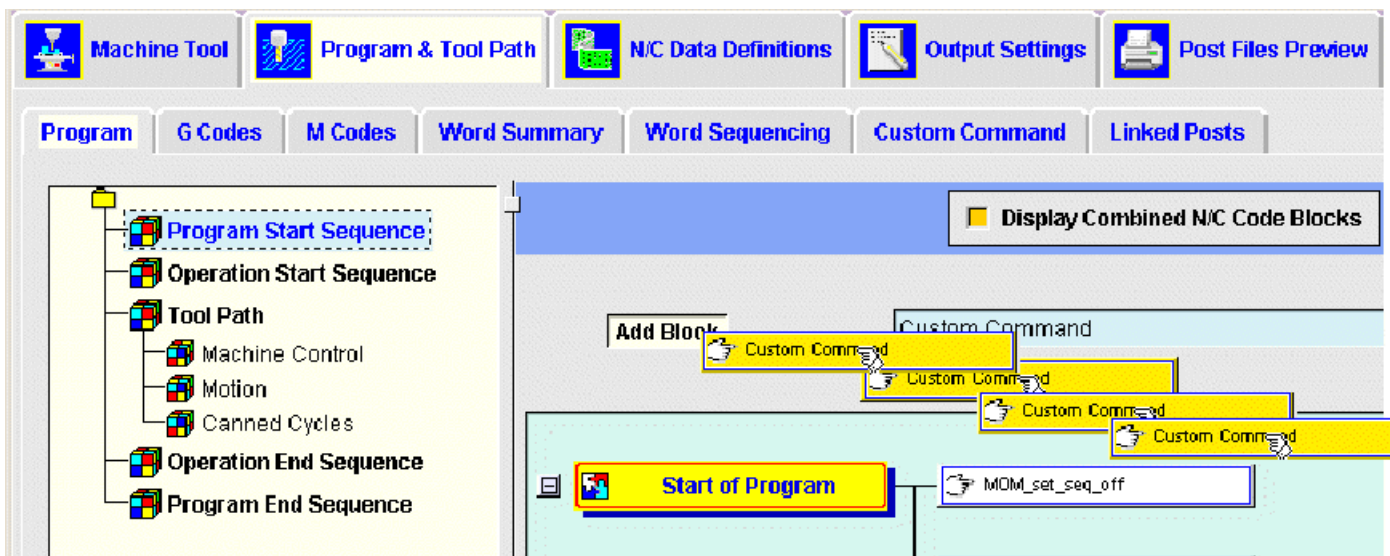
Another method of adding a custom command is by use of the **Program** tab of the **Program and Tool Path** property page **Add Block** feature. With this method you are able to cut and paste the contents of an existing text file into the custom command block.



- If necessary, select the **Program and Tool Path** property page, and then select the **Program** tab.

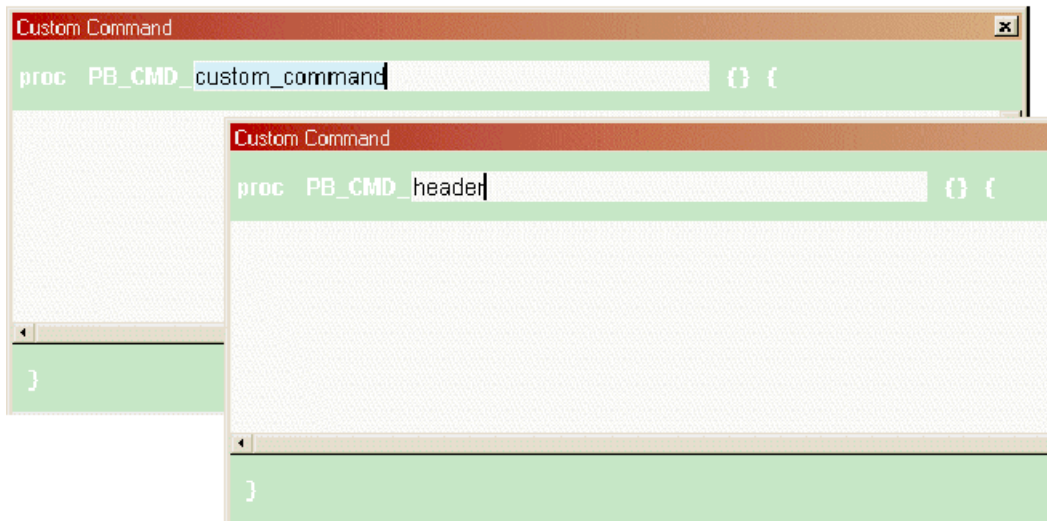
Since the block of information desired is at the beginning of the program, you will want to insert the custom command at the beginning of the Program Start Sequence.

- If necessary, highlight the **Program Start Sequence**, select **Custom Command** from the pull down menu and then select the **Add Block** button.
- Drag and then drop the **Custom Command** block prior to the **MOM\_set\_seq\_off** block.



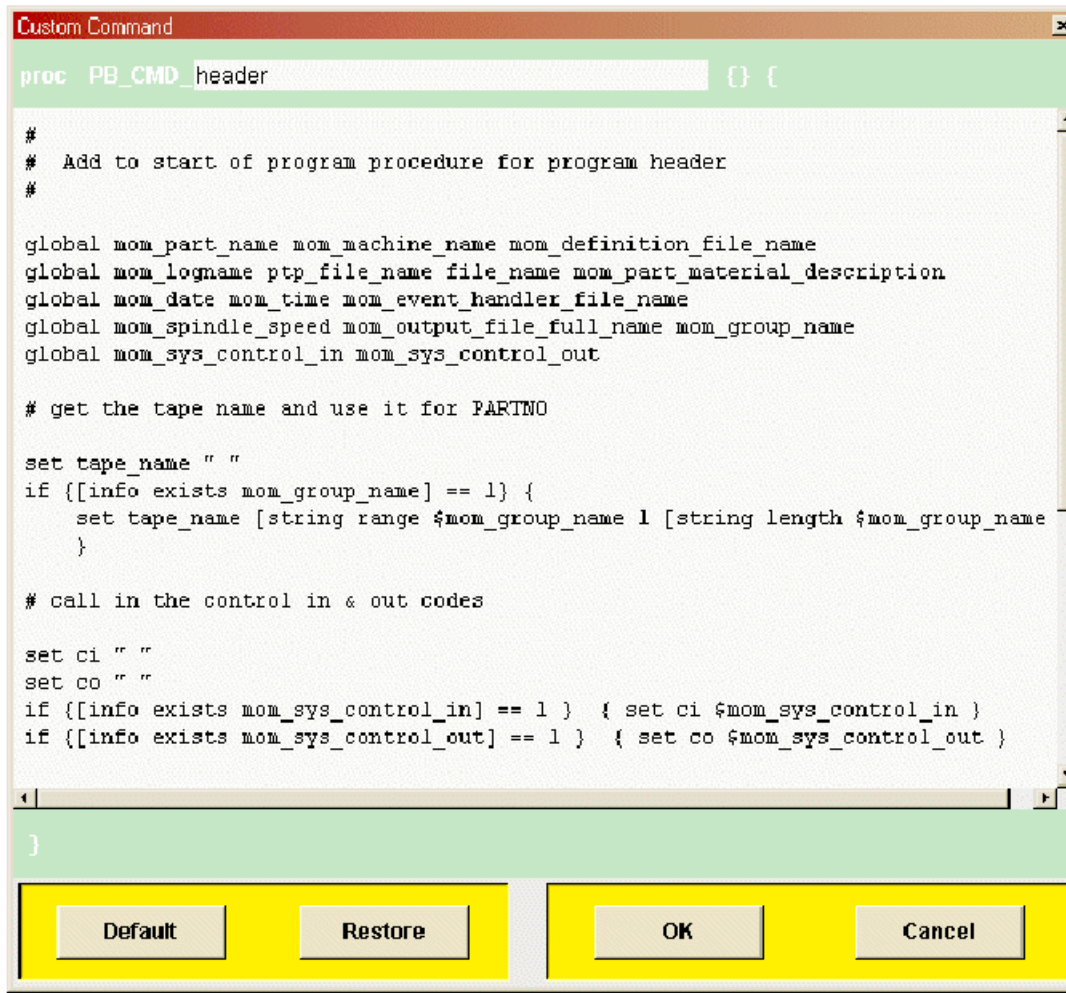
The Custom Command dialog is displayed.

- ☐ Rename the Custom Command to **header** by highlighting **custom\_command** in the proc name box and typing **header**.



- Step 4:** Select the data for the Custom Command from the file **custom\_command\_1.txt** from the Student Parts directory.
- ☐ In XP Explorer, navigate to the student parts directory, double-click on the file **custom\_command\_1.txt**, select the entire contents of the file and then select Ctrl-C to copy.
- Step 5:** Insert the text into the Custom Command.

- ☐ Right-click ® **Paste** the contents of the text file into the Parameter Window.



The contents of the paste buffer is inserted into the Parameter Window.

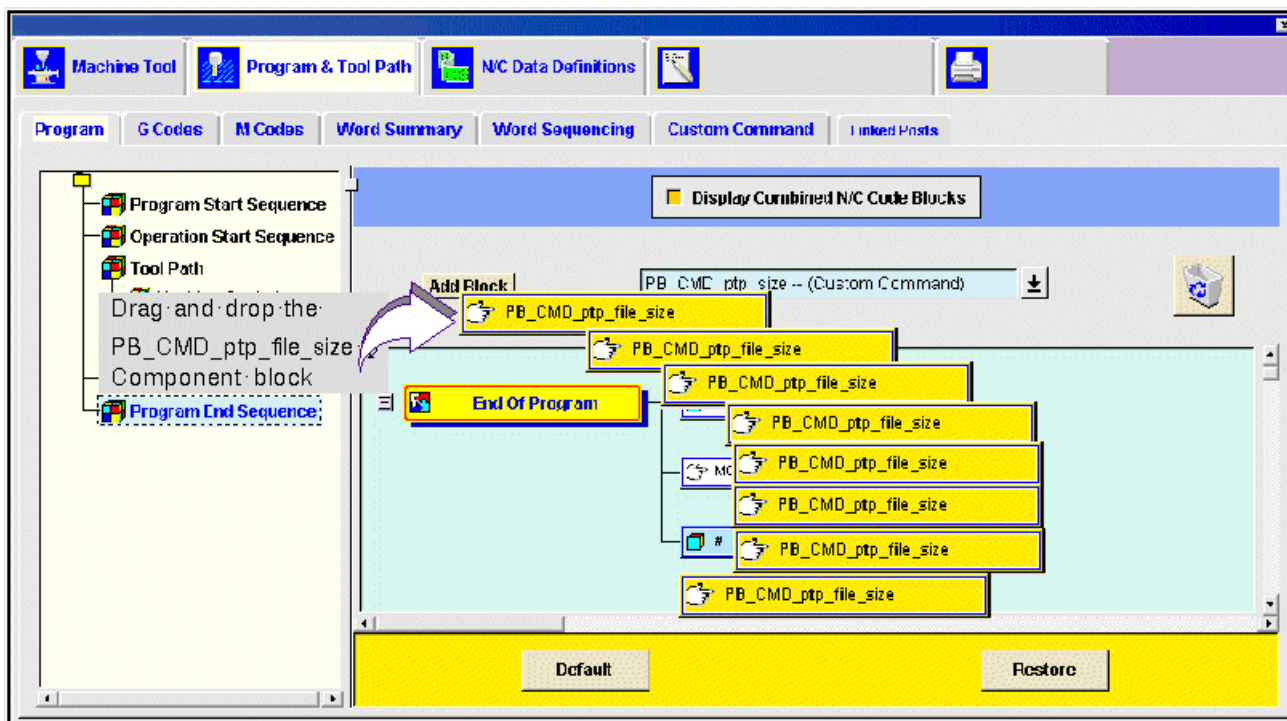
- ☐ Click **OK**.

You have just created a Custom Command that will calculate the physical file size and another that creates header information, using two different methods of creation. The header information has already been placed in the proper location for output. The Custom Command that calculates the physical file size needs to be placed at the end of the program for output.

**Step 6:** Add the previously created Custom Command for specifying file size to your post processor.

You will now place the Custom Command used to calculate the file size of the output file at the end of the program (note that if you placed this Custom Command before or after a previous block component, the calculation of the file size of the output file would not be accurate since it is not located after the last block component).

- ☐ Select **Program End Sequence** in the component window.
- ☐ Select **PB\_CMD\_ptp\_file\_size**, from the **Add Block Component Available** option menu, drag and drop the command **after** the last block component.



You have just inserted the Custom Command for calculating file size into the proper location of your post processor. You will now verify those commands by running the post processor against your test part and then verifying the results.

**Step 7:** Save the post and run it against the test part, mill\_test.

- ☐ If necessary, enter the Manufacturing Application in NX.
- ☐ Expand the **T12345-A** and click **FACE\_MILLING**.



- ☐ Select Post Process .

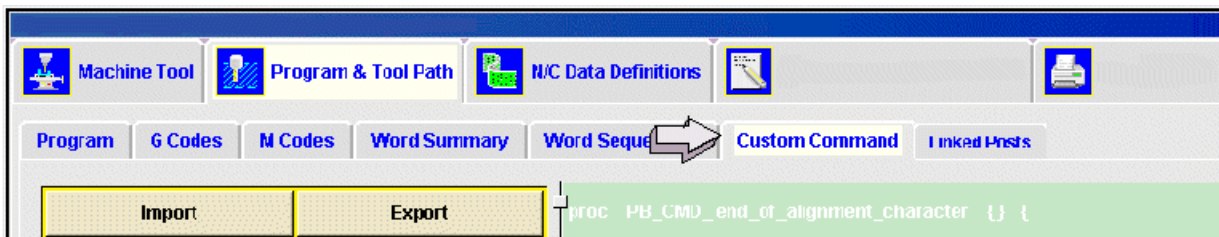
- ☐ Click or browse to your post processor **\*\*\*\_my\_post** and click **OK**.
- ☐ Accept the default for Output File and then click **OK**.
- ☐ Verify the output.

Your output should be similar to the following:

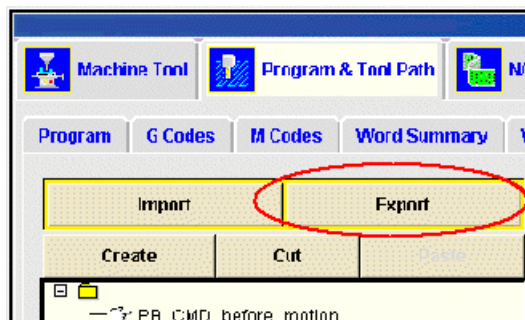
```
=====
PARTWO
(PART NAME : PBT_MILL_TEST.PRT)
(CREATED BY : )
(CREATION DATE : )
(CREATION TIME : )
(UGPOST NAME : ***_MY_POST)
(OUTPUT FILE : .....PBT_MILL_TEST.PTP)
%
N0010 G40 G17 G90 G70
N0020 G91 G28 Z0.0
:0030 T30 M06
N0040 G00 G90 X19.5 Y-2 S456 M03
N0050 G43 Z4. H30
N0060 Z2.2
N0070 Z2.1
N0080 G01 Z2. F13. M08
N0090 X17.
:
N0350 G01 Z2.
N0360 X17.
N0370 X-3.
N0380 G00 X-5.25
N0390 Z4.
N0400 Z7.
N0410 X-1. Y4.
N0420 Z20.
N0430 X-10. Y-10.
N0440 M02
%
[PTP file size = 969 bytes 0.1 feet]
```

**Step 8:** Export the Custom Command that created header information as a file to the Custom Command directory.

- ☐ Click **Custom Command**.



- ☐ Click **Export**.

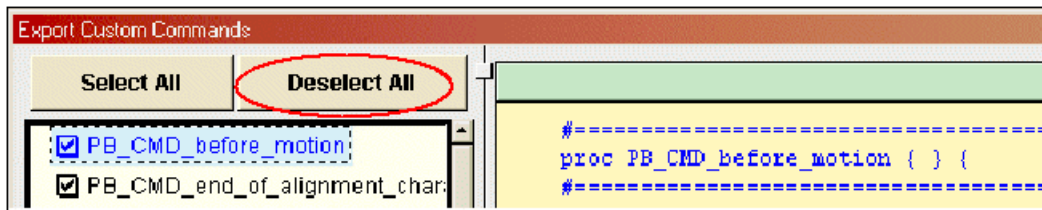




The Export Custom Commands dialog is displayed.

You will notice that all Custom Commands listed in the parameter window are selected (checked). You only want to export the **PB\_CMD\_header** custom command. You will therefore want to **Deselect All** and then select just the **PB\_CMD\_header** command.

- ☐ Click **Deselect All**.

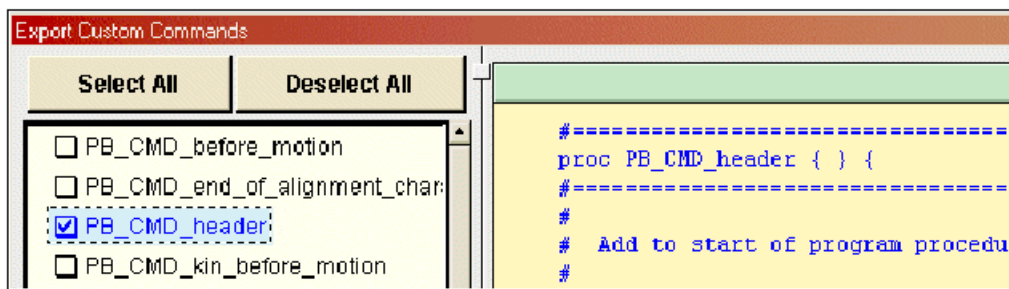


All Custom Commands are deselected.

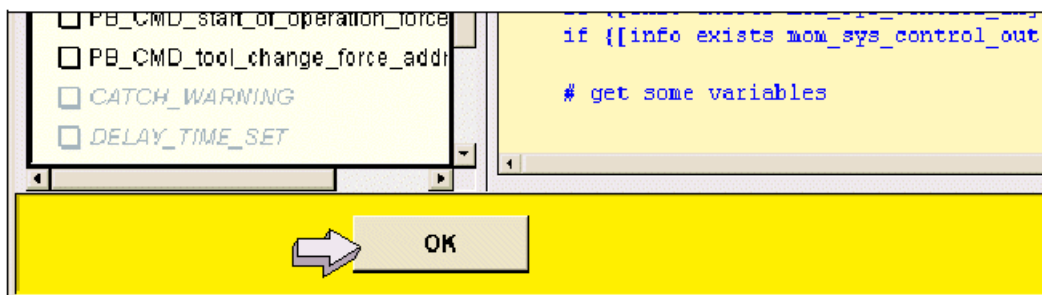
You will now select the **PB\_CMD\_header** Custom Command for Export.

- ☐ Double-click, **PB\_CMD\_header**.

The **PB\_CMD\_header** Custom Command is selected.

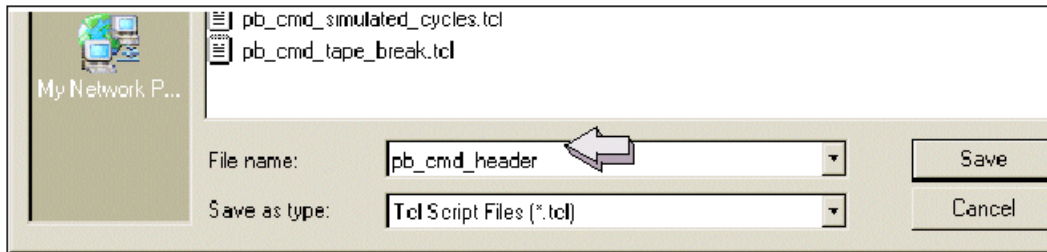


- ☐ Click **OK**.



The **Select a file** dialog is displayed.

- ☐ Type **pb\_cmd\_header** in the File name box.



- ☐ Click **Save**.

The Custom Command, **pb\_cmd\_header** is saved in the Custom Command directory and can be used when creating other post processors.

This completes the activity and the lesson.

## Summary

The Custom Command feature of the Post Builder provides an easy method for the use of pre-defined or user created procedures that allow for custom output. Custom Commands can be:

- Retrieved from a NX provided Custom Command library.
- Created to meet individual machine tool custom output.
- Created, modified and saved for use in the building of other post processors.



## Lesson

# 9 *User Defined Events (UDEs)*

### **Purpose**

This lesson describes the function, modification and creation of User Defined Events.

### **Objective**

Upon completion of this lesson, you will be able to:

- Create, modify and use User Defined Events (UDEs).

## User Defined Events

User Defined Events (UDEs) provide a mechanism of having certain machine functions become user definable. This may include spindle speed and direction, coolant on or off, and clamping of the work piece. Numerous User Defined Events are supplied as standard with NX.

UDEs can be specified at the **start of path, end of path, in path** or can be attached to any CAM parent object (Program, Method, Machine Tool, Geometry) using the User Defined Events dialog.

UDEs generate Events with all Event data being passed to the post processor as a MOM event and as mom variables.

The description of the UDEs and the data associated with them are defined in the file **ude.cdl** located in the **UGII\_BASE\_DIR/mach/resource/user\_def\_event** directory. This file is referenced by the configuration file. Parameters that are set in the UDE are converted to the corresponding mom variable. A corresponding procedure in the post processor Tcl file is a requirement for output. UDEs can be included in the definition file.

## User Defined Event syntax

```

EVENT <name>
{
  POST_EVENT <post_name>
  UI_LABEL <ui_name>
  CLASS Type, subtype, subtype; Type, subtype, subtype;
  CATEGORY Mill | Drill | Lathe | WEDM
  PARAM <param_name1>
  {
    TYPE b | i | d | s | o | p | v
    DEFVAL <default value as string>
    TOGGLE Off | On
    OPTIONS <string1, string2, string3>
    UI_LABEL <ui_name>
  }
  PARAM <param_name2>
  PARAM <param_name3>
}

```

**EVENT** <name> defines the Event "name".

**POST\_EVENT** <post\_name> specifies that the Event "name" will be output to the post processor as an Event called "MOM\_post\_name". This attribute of the EVENT is optional and if not specified, the Event will be output to the post processor as an Event called "name".

**UI\_NAME** <ui\_label> specifies that "ui\_name" is a string used for the title of the dialog. This attribute of the EVENT is optional and if not specified the Event name itself is used as the "ui\_name".

**CATEGORY** attribute specifies the machining type (mill, lathe, WEDM, etc.) in which the Event is relevant. This attribute is optional. The Event can be specified to be relevant in more than one machining mode. When this attribute is not specified the Event becomes relevant in all machining modes.

**PARAM** <param\_1> specifies a parameter of the Event. The parameter has a name "mom\_param\_name1".

**TYPE** is an attribute of the PARAM and specifies the parameter type. The possible values for TYPE are:

- b - binary
- i - an integer
- d - a real number
- s - a string
- o - one of a possible set of options
- p - a point
- v - vector

**DEFVAL** <defval> specifies the default value of the parameter. This is an optional attribute (TYPE o) and if not specified the following will be the defaults according to the TYPE of the parameter:

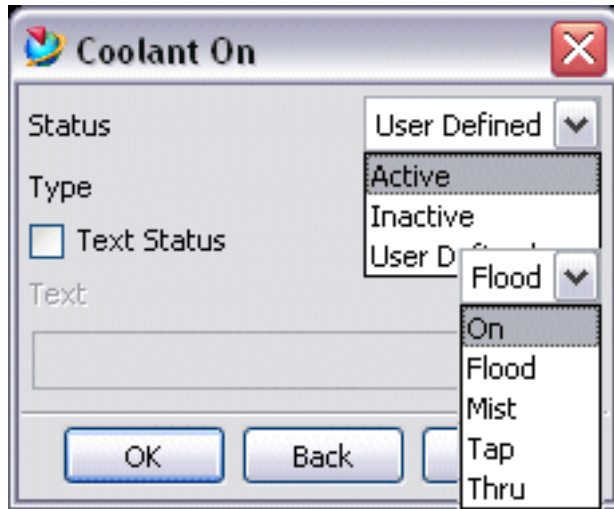
- b - the value can be 0 or 1
- i - default value is 0
- d - default value is 0.0
- s - default value is an empty string
- o - default value is the first one of the list of options provided
- p - default value does not apply
- v - vector of 0, 0, 1

**TOGGLE** is an attribute of the parameter which specifies a default value, the default value is either On or Off. The TOGGLE attribute is optional and if not specified the parameter is mandatory.

**OPTIONS** is a required attribute of the parameter when the TYPE is 'o'. This provides all the possible values that the parameter can have. The values are specified as comma separated strings.

**UI\_LABEL** <ui\_label> specifies that "ui\_name" will be the text displayed in the dialog presented. This is an optional attribute and if not specified the "param\_name1" will be used as the "ui\_name".

The following illustrates the Tool Change UDE with corresponding dialog.



```

EVENT coolant
{
  POST_EVENT "coolant_on"
  UI_LABEL "Coolant On"
  CATEGORY MILL DRILL LATHE
  PARAM command_status
  {
    TYPE o
    DEFVAL "Active"
    OPTIONS "Active","Inactive","User Defined"
    UI_LABEL "Status"
  }
  PARAM coolant_mode
  {
    TYPE o
    DEFVAL "Flood"
    OPTIONS "On","Flood","Mist","Tap","Thru"
    UI_LABEL "Type"
  }
  PARAM coolant_text
  {
    TYPE s
    TOGGLE Off
    UI_LABEL "Text"
  }
}

```

## Activity — Create user defined events

In this activity you will create a generic post processor used for testing purposes in your home directory using the Post Builder. You will then create and test various User Defined Events.

**Step 1:** Create the post processor for testing purposes.

- ☐ If necessary, activate the Post Builder.
- ☐ Open the **\*\*\*\_my\_post** post processor located in your postprocessor directory.

**Step 2:** Change the User Defined Event for coolant from the default of Flood to Mist.

- ☐ Copy the **ude.cdl** file and rename to **ude.cdl\_org**.
- ☐ Edit the **ude.cdl** file, located in your home **/resource/user\_def\_event** directory, locate the **Event Coolant**, and change the current default value for the parameter **coolant\_mode** from **Flood** to **Mist**.

```
PARAM coolant_mode
{
    TYPE o
    DEFVAL "Mist"
    OPTIONS "On","Flood","Mist","Tap","Thru"
    UI_LABEL "Type"
}
```

**Step 3:** Create a new User Defined Event to Load a new coordinate system to output a G59 code.

- ☐ Add the following lines of code at the end of ude.cdl file, following the

```
#
# Event for coordinate offset
#
EVENT sys_offset
{
  POST_EVENT "csys_load"
  UI_LABEL "Coordinate Offset"
  PARAM command_status
  {
    TYPE o
    DEFVAL "Inactive"
    OPTIONS "Active","Inactive"
    UI_LABEL "Offset"
  }
  PARAM csys_register
  {
    TYPE o
    DEFVAL "54"
    OPTIONS "54","55","56","57","58","59"
    UI_LABEL "Coordinate System"
  }
}
#
#End of coordinate system event
#
```

- ☐ Save the file.

**Step 4:** Reload the revised ude.cdl file

- ☐ From the NX tool bar, select **Preferences**→**Manufacturing**.
- ☐ Select the Configuration property page; under Configuration File, select the **Reset** button, then choose **OK** to reload the **ude.cdl** file.

If the contents of the **ude.cdl** file are correct, the file will reload with no error messages. If error messages appear, check to make sure the syntax is as specified previously, fix the errors and repeat the above process.

**Step 5:** Add a Custom Command to the \*\*\*\_my\_post post processor to utilize the newly created User Defined Event.

Using the Custom Command feature of the Post Builder, add the Custom Command **PB\_CMD\_coordinate\_offset** to the operation Start Sequence.

- ☐ If necessary, using the Post Builder, open the file **\*\*\*\_my\_post**.
- ☐ Click **Program and Tool**, then select **Program**.

- ☐ Click **Program Start Sequence**, in the components window, select the pull down menu and select the **Custom Command** block, drag and drop **Custom Command** block as the first item in **Start of Program**.
- ☐ In the Custom Command window, change the name from Custom Command, to **coordinate\_offset**.
- ☐ Add the following text to the Custom Command:

```
#
# custom command to output a G54 thru G59 coordinate
# offset
#
uplevel #0 {
proc MOM_csys_load { } {
global mom_csys_register
MOM_output_literal "G$mom_csys_register"
}
}
```



- ☐ **Save** the post processor.

**Step 6:** Use the newly created User Defined Event.

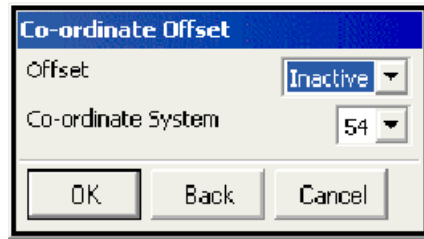
- ☐ If necessary, open the test part file, **mill\_test**.
- ☐ Change the **Operation Navigator** view to the **Geometry** view.
- ☐ Right click **MCS\_MILL** , select **OBJECT**→**Start Event**.

The User Defined Events dialog is displayed.

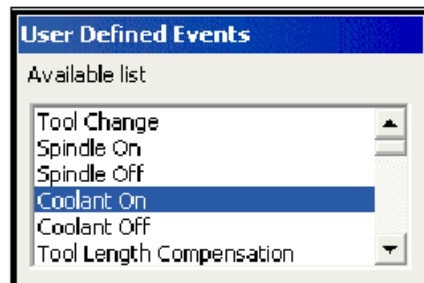
- ☐ Select **Coordinate Offset** from the Available items list, then choose **Add**.



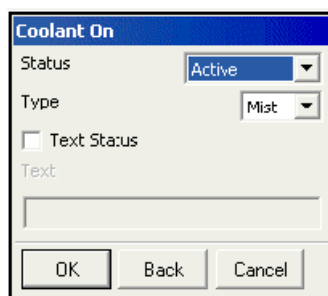
The Co-ordinate Offset dialog is displayed.



- ☐ Activate the Event, by selecting **Active** from the **Offset** list and then select **59** from the **Co-ordinate System** list.
- ☐ Choose **OK** from the **Co-ordinate Offset** dialog.
- ☐ Double-click on the **Coolant On** command from the User Defined Events dialog.



Notice the default has changed to **Mist** from **Flood** (this was changed in Step 3).



- ☐ Click **Cancel** from the Coolant On dialog.
- ☐ Click **OK** from the User Defined Events dialog.
- ☐ Highlight and post the **FACE\_MILLING** operation, using the post processor created in Step 1.

Name	Order Group	Method	Geometry	Tool
NC_PROGRAM				
NONE				
POST_DEBUG				
FACE_MILLING	POST_DEBUG	MILL_ROUGH	MILL_GEOM	FACEMILL_4.00
PLANAR_PROFILE_RGH	POST_DEBUG	MILL_ROUGH	MILL_GEOM	MILL_1.00
PLANAR_PROFILE_FIN	POST_DEBUG	MILL_FINISH	MILL_GEOM	MILL_1.00
PLANAR_MILL	POST_DEBUG	MILL_FINISH	MILL_GEOM	BALLMILL_.875
DRILLING	POST_DEBUG	DRILL_500_TA...	DRILL_GEOM	DRILL_.500
DRILLING_1.75	POST_DEBUG	DRILL_500_TA...	DRILL_GEOM_BORE	DRILL_1.75
BORE_1.000	POST_DEBUG	BORING	DRILL_GEOM_BORE	BORING_BAR...
TAP_5-13UNC	POST_DEBUG	TAPPING	DRILL_GEOM	TAP.5-13UNC

- ☐ Verify the output (check for the G59 code).

```
=====
FACE_MILLING OPERATION
=====
#
N10 G40 G17 G91 G90 G70
N20 G59
N30 G91 G28 Z0.0
=====
N220 Y1.7766
N230 X7.
N240 Y3.5766
N250 X3.5766
N260 Y4.4234
N270 X10.4234
N280 Y3.5766
N290 X7.
N300 G00 Z4.
N310 Z7.
N320 X-1. Y4.
N330 Z20.
N340 X-10. Y-10.
N350 M02
```

This ends the activity.

## Activity — Modify the coolant UDE for thru-spindle

In this activity, you will modify the existing coolant UDE in the `ude.cdl` file to allow coolant thru the spindle. You will then use this modified UDE with the previously created M-code group, created in Lesson 2, to enable thru the spindle coolant.

**Step 1:** Modify the existing coolant UDE to allow coolant thru the spindle.

- ☐ Using a text editor, open the **ude.cdl** file located in the **/resource/user\_def\_event** directory.
- ☐ Locate the coolant Event (similar to the following):

```
EVENT coolant
{
  POST_EVENT "coolant_on"
  UI_LABEL "Coolant On"
  CATEGORY MILL DRILL LATHE
  PARAM command_status
  {
    TYPE o
    DEFVAL "Active"
    OPTIONS "Active","Inactive","User Defined"
    UI_LABEL "Status"
  }
  PARAM coolant_mode
  {
    TYPE o
    DEFVAL "Flood"
    OPTIONS "On","Flood","Mist","Tap","Thru"
    UI_LABEL "Type"
  }
  PARAM coolant_text
  {
    TYPE s
    TOGGLE Off
    UI_LABEL "Text"
  }
}
```

- ☐ Add a **"High"** option to the **coolant\_mode** variable.

```
PARAM coolant_mode
{
  TYPE o
  DEFVAL "Flood"
  OPTIONS "On","Flood","Mist","Tap","Thru","High"
  UI_LABEL "Type"
}
```

- ☐ Save the **ude.cdl** file.

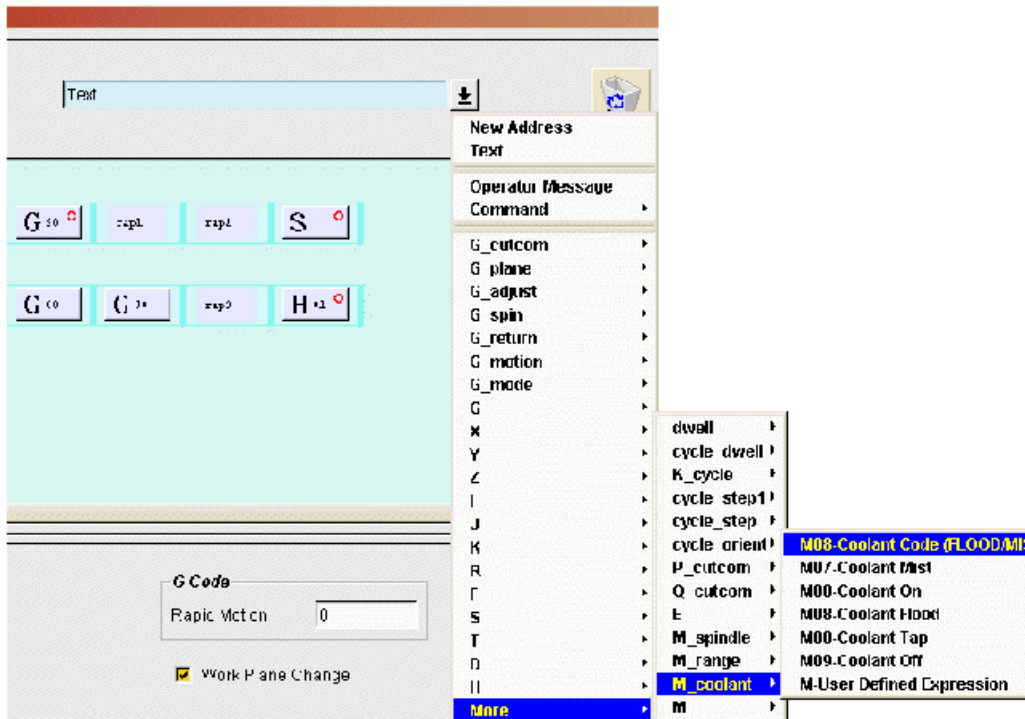
**Step 2:** Modify the post processor to recognize the UDE coolant-thru.

- ☐ If necessary, in Post Builder, open the **\*\*\*\_my\_post** post processor.

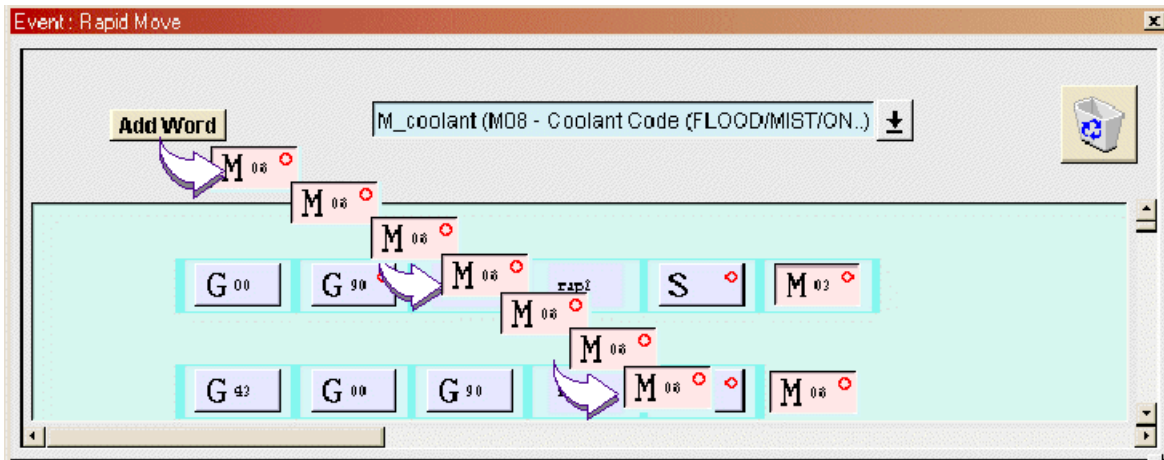
- ☐ Click the **Program and Tool** path property page.
- ☐ Click **Program**.
- ☐ Highlight the **Motion** sequence.
- ☐ Select the **Rapid Move** box.

The Event Rapid Move dialog is displayed. You will now insert a block just **after** the Rapid Move event to load the coolant M code. The new block will output with the "Z" axis rapid move.

- ☐ Next to the **Add Word** button, from the pull down menu choose **More ® M\_coolant ® M08 Coolant Code (FLOOD/MIST/ON...)** as the item to add.



- ☐ Drag the **Add Word** button down and release it just below the last motion block.



- ☐ Choose **OK**.

**Step 3:** Create a Custom Command, coolant\_thru which will add the coolant thru the spindle.

You will now create a custom command for coolant thru the spindle.

- ☐ If necessary select the **Program** tab.
- ☐ Select **Program Start Sequence**.
- ☐ Select the pull down menu and select **Custom Command** option.
- ☐ Left click the **Add block** drag and place the Custom Command as the first item in Start of Program.
- ☐ After the Custom Command dialog opens, change the phrase Custom\_Command to Coolant\_thru.



Add the following text to the Custom Command:

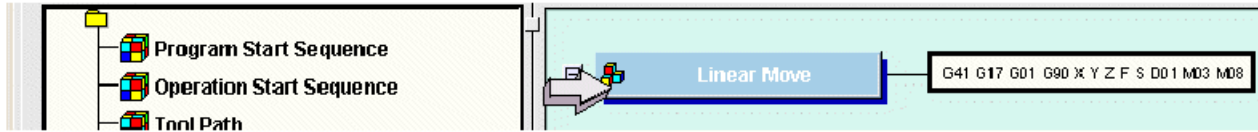
```
uplevel #0 {
  set mom_sys_coolant_code(HIGH) "29" }
```

You have just created the custom command for high pressure coolant thru the spindle.

**Step 4:** Add the newly created coolant 'M' code to the linear move event.

- ☐ Choose the **Program** tab and then **Motion** under the **Program and Tool Path** event.

- ☐ Click on the Linear Move block.



The Event: Linear Move dialog is displayed.

- ☐ Right click on the **M\_coolant** address, select **change element**, then select **M08-Coolant Code (FLOOD/MIST/ON..)**.
- ☐ Click **OK** from the Event: Linear Move dialog.
- ☐ **Save** the post processor.
- ☐ Test the post processor against a current operation.

When testing, be sure to test against all possible conditions (i.e. coolant UDE not active, coolant UDE active but not set to coolant thru, and so on). Verify that the coolant is turned off at the end of the operation.

In summary:

- ☐ The programmer chose high pressure thru the spindle by turning on the UDE which you just modified.
- ☐ The Custom Command will add an additional M code for coolant high.
- ☐ The "coolant\_on" block outputs the M coolant word which is set to a value of "29".

You could have also designed a Custom Command that would obtain the actual coolant value from the UDE or allow the programmer to specify which coolant code was desired. It would also be possible to determine from the type of tool, if coolant thru capability was allowed, and if it were to automatically insert the proper code.

This concludes the activity.

## Summary

User Defined Events provide a mechanism of allowing Machine Control Events to be defined by the user. Machine Control Events (considered to be UDEs) are used to convey information concerning the physical state of the machine tool, such as spindle speed, coolant status, etc., to the post processor. User Defined Events are:

- Used to describe the format of all APT style post processor commands.
- Specified at the start or end of path or can be attached to any CAM object.
- Defined in the ude.cdl file that is referenced by the configuration file.





## Lesson

# *10 Virtual NC Controller*

### **Purpose**

In this lesson, you will learn how to create a Virtual NC Controller, used by the Integrated Simulation and Verification module for tool path simulation.

### **Objective**

Upon completion of this lesson, you will be able to:

- Create a Virtual NC Controller.

## Integrated Simulation and Verification overview

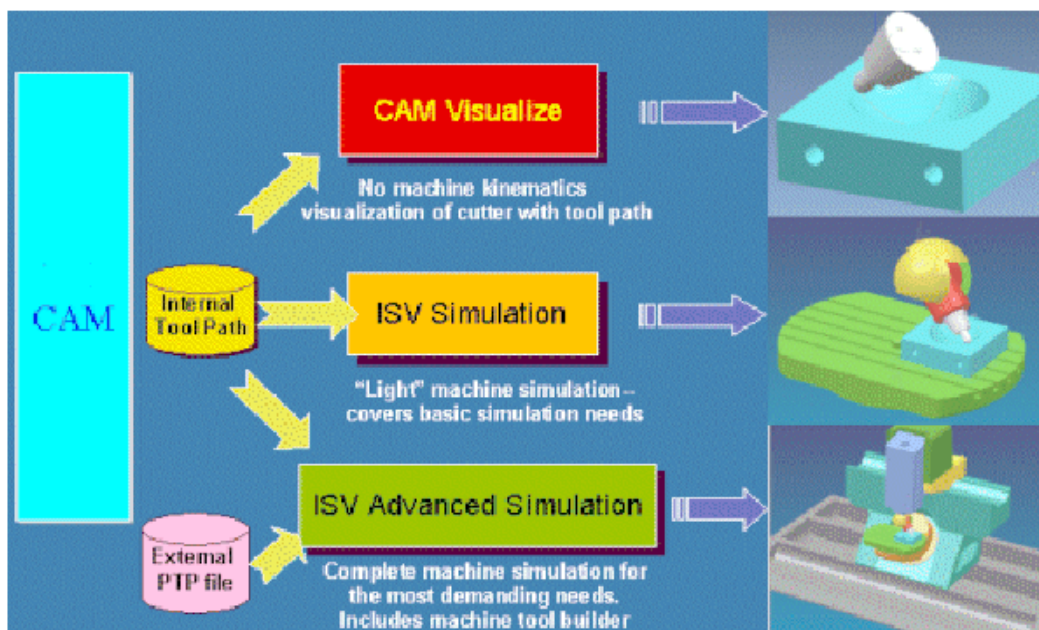
The Integrated Simulation and Verification module (IS&V) allows the simulation of a machine tool with an actual piece part. The simulation process animates the exact machine tool motions, taking into account controller functions and cutting tool configurations.

IS&V features collision checking, allowing collision detection between machine components, fixtures, tools, parts and the in-process work piece. You may also view machine controller functions such as macros, subroutine calls, cycles and function M, G and H commands.

IS&V improves the quality of machining processes by allowing the comparison of the designed part to the part which is being manufactured. Reduction in cost can be obtained by the elimination of expensive and time consuming dry runs; reduced manual operator intervention; and the reduced risk of expensive damage to machine tools, fixtures and parts by elimination of collisions.

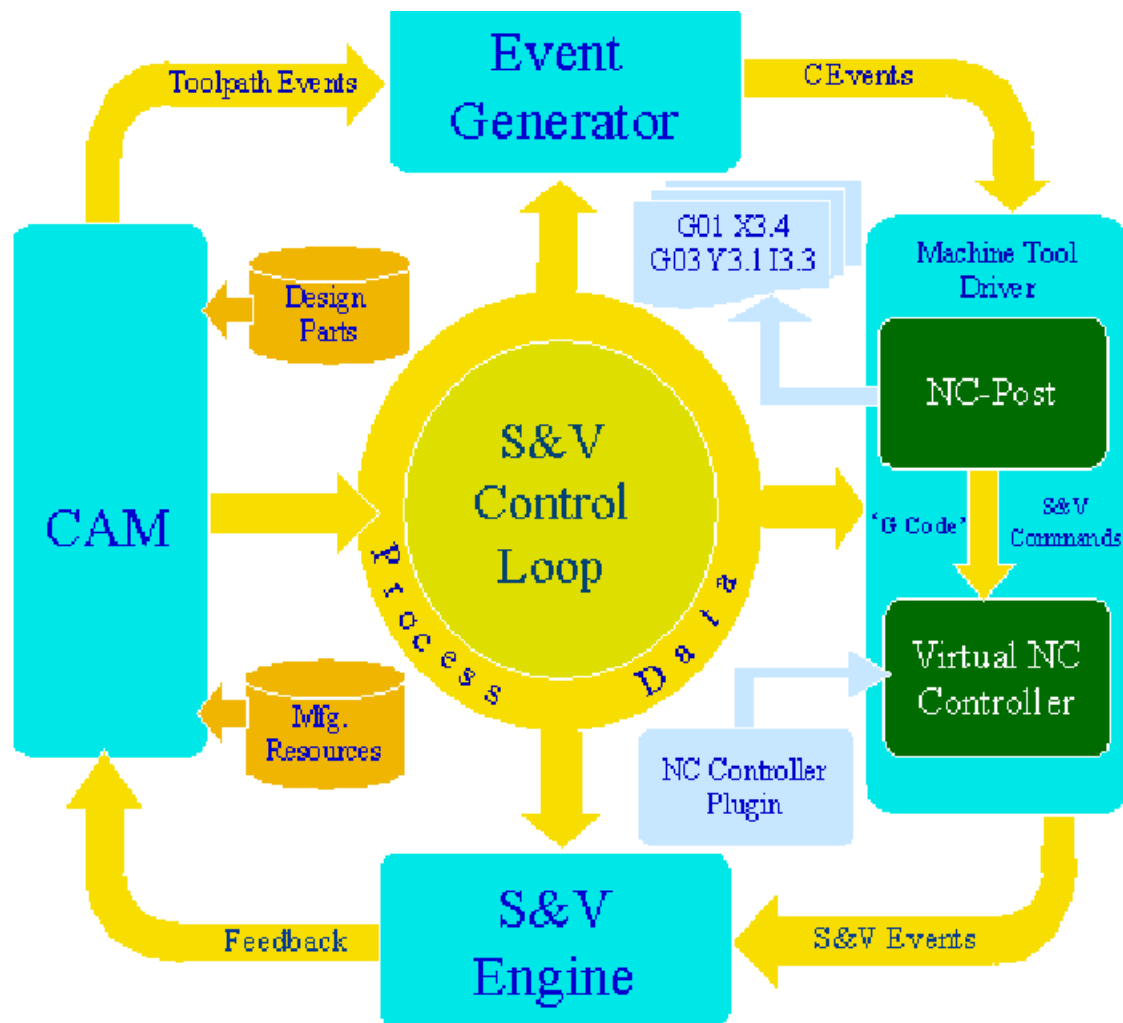
IS&V consists of the following components:

- Visualize
  - Simulation
  - Advanced Simulation
  - Machine Tool Builder
  - Machine Tool Driver
  - Setup Configurator



## Machine Tool Driver

- Generates motion control program and emulates the CNC controller
  - Accurate path based on machine tool configuration
  - Handles specific machine tool features including macros, cycles and subroutines
  - Can be customized using Tcl scripting language
  - Text and graphics feedback initiated by Events



The **Machine Tool Driver (MTD)**, also referred to as the Virtual NC controller, creates the CNC program that emulates the CNC controller. This is a programmable interface that instructs the machine tool model on actual movements and how those movements are displayed. Any motion and feedback displayed during machine tool simulation is controlled by the dedicated **MTD**.

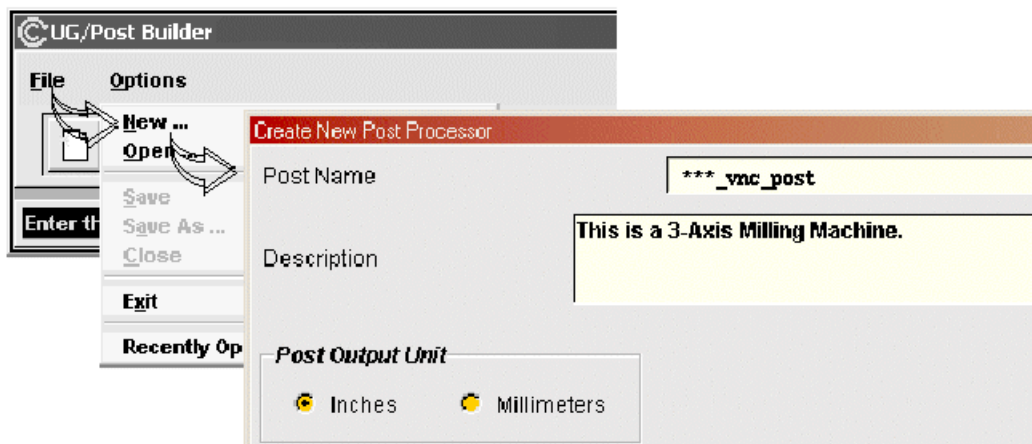
For comparison purposes, the **MTD** is analogous to the machine tool simulator as the CNC controller is analogous to the machine tool that it controls. For each machine tool in the machine tool library, there is a **MTD** driver available. For creating an **MTD** for a new machine tool, you can modify a generic driver to work with that machine or use the Post Builder to generate the Virtual NC controller (MTD and Virtual NC controller refer to the same object). **MTD's** are written in the Tcl scripting language but may also be developed in higher level languages such as C++. **MTD's** can emulate special cycles, User Defined Events (UDE's), macros and other CNC controller dependent functions that the Manufacturing application does not support.

## Activity — Use Post Builder to create a VNC

In this activity you will use the Post Builder to create a Virtual NC controller used with the Integrated Simulation and Verification module.

**Step 1:** Start the Post Builder.

- ☐ On the desktop menu bar, choose **Start® All Programs NX→PostTools® Post Builder**.
- ☐ Select **File→New** and in the **Name** field, name the post processor **\*\*\*\_vnc\_post**, where **\*\*\*** stands for your initials.

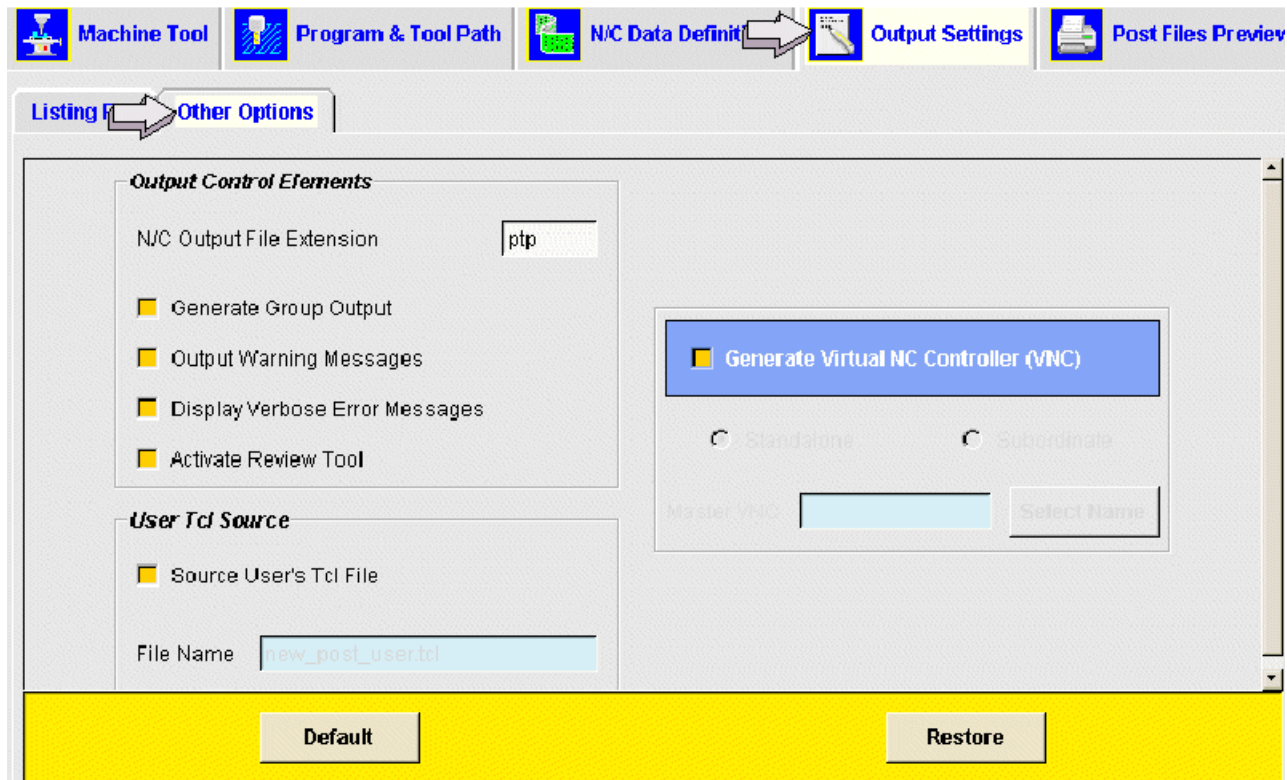


**Step 2:** Create a generic 3-axis milling post processor.

- ☐ For **Post Output Unit** you will accept the default of **inches**.  
For **Machine Tool** you will accept the default of **Mill**.  
You will accept the default for 3-axis.  
For **Controller** you will accept the default of **Generic**.
- ☐ Select **OK**.

**Step 3:** Set the Virtual CNC Controller options.

- ☐ Select the **Output Settings** tab, then choose the **Other Options** tab.



- ☐ Turn On the Generate Virtual NC Controller item.

You may designate that the current post is a standalone or master of a linked post or specify it as a subordinate post of a linked post.

You must now edit the Custom Command that controls the name of motion axes to match the name assigned to the axes designated by the machine tool builder.

- ☐ Select the Program and Tool Path tab, then select the Custom Command tab.
- ☐ Highlight PB\_CMD\_vnc\_\_\_\_map\_machine\_tool\_axes.
- ☐ In the Custom Command window, scroll to the **set mom\_sim\_mt\_axis(X)** **X**, this name must match the axis name specified in the machine tool builder.
- ☐ Save your post processor.

Files created for the Virtual NC controller will have a **VNC\_** prefix and a **.tcl** extension. As you can see, it is very easy to create the Virtual NC controller by just turning on the option.

## Summary

The Integrated Simulation and Verification module (IS&V) allows you to simulate a machine tool with an actual piece part, giving you an overview of the entire machining process. The simulation process animates the exact machine tool motions, taking into account controller functions and cutting tool configurations.

In this lesson you:

- Reviewed the components that comprise the Integrated Simulation and Verification module.
- Used the Post Builder to generate a Virtual NC Controller used in machine tool simulation.





## Lesson

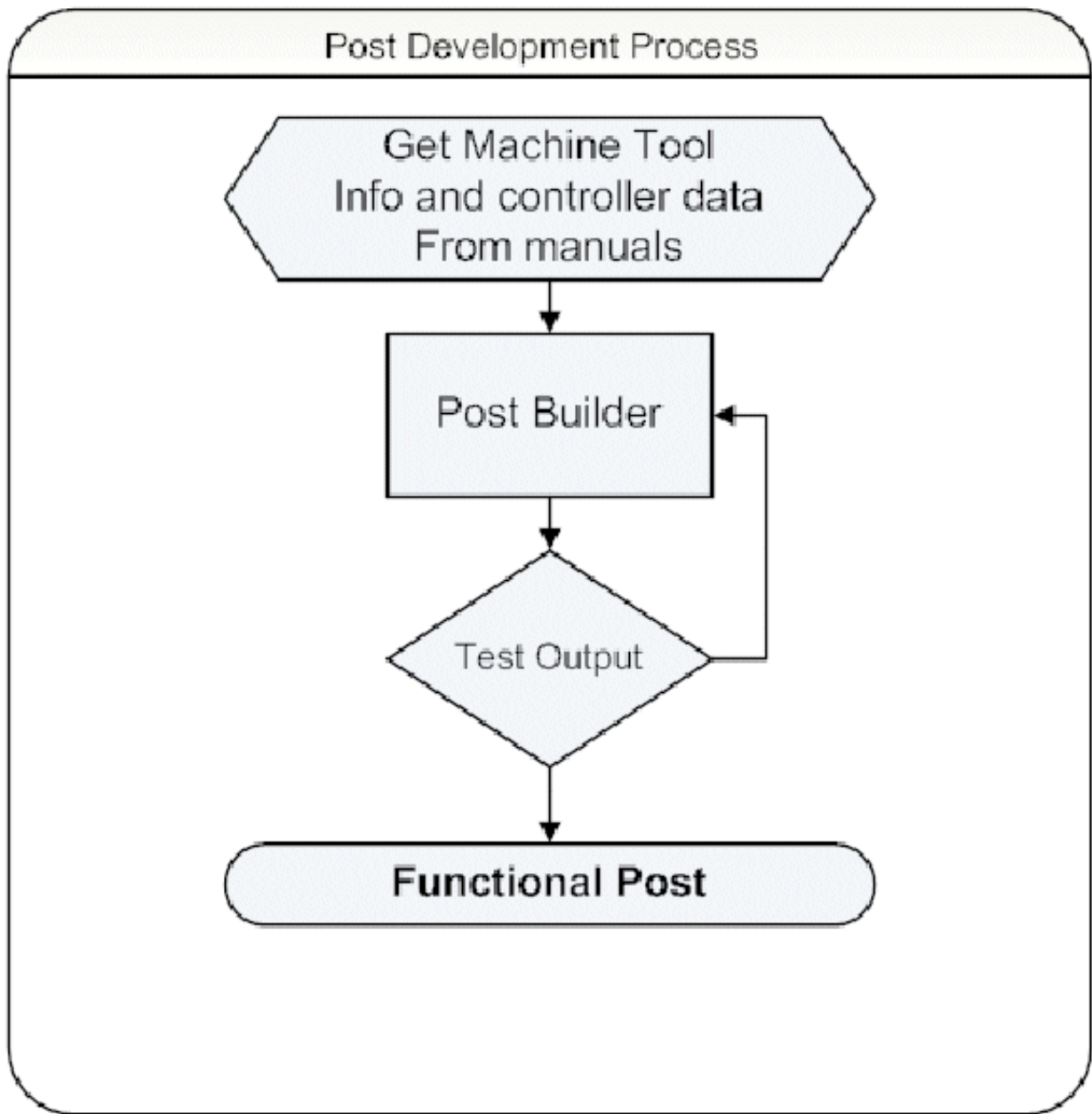
# *11 A guide to best practices of building a Post Processor*

## Overview

As you gain experience with the various Post components, you will find yourself experimenting with numerous methods and techniques as you go through the post development cycle. Eventually, you will settle on methods that you have become familiar and comfortable with.

The following suggestions and procedures will aid you in the creation of post processors.

- Use the following flow chart as a tool to decide if a post processor can be created through **Post Builder**, by creating and modifying Event Handlers and Definition files or a combination of both.



- When adding multiple **custom commands**, add and test each custom command individually. This will make the process of debugging easier and less confusing when you encounter Tcl syntax or Post errors.
- Use the **Review Tool** always when debugging a post processor. Although the Review Tool increases the time required for processing, it is an invaluable aid in the debugging process. The Review Tool may also be used to review MOM Events, MOM variables per Event, find missing procedures and find MOM variable names.
- Always make a copy of the main CAM **mach** directory for testing and debugging purposes. The idea behind this concept is that you can experiment with various ideas and techniques without the worry of corrupting any system files that would normally be used by NX/Post or the Post Builder.
- If using the Post Builder to create a post and you find that you may need to edit the Event Handler or Definition file, make sure that you perform all edits and modifications that you can do within the Post Builder. Once you "manually" edit the files, which Post Builder creates, the link between those files and Post Builder are lost. This prevents you from using the Post Builder on that particular post again.
- Use **ugpost\_base.tcl** to define common G & M codes, common addresses, frequently called procedures and common routines.
- UDE defined variables can be retrieved in MOM\_start\_of\_program or MOM\_start\_of\_path by use of Tcl command, **uplevel**.
- Manufacturing attributes **cannot** be retrieved in MOM\_start\_of\_program.
- Expression variables can be retrieved at any time.
- Additional Tcl or Tk programs can be sourced and run from within the machine tool Tcl file.
- To remove a file in MOM use **MOM\_remove\_file** .
- Post Builder is independent of the core release of NX. You can take the latest version of the Post Builder and install it independently of the release.



## Appendix

# A Custom command examples

A

### Examples of some useful Custom Commands

```
=====

# Add the following in a custom command procedure called ATTRIBUTE
# The following lines allow the use of global variables in this procedure

global mom_tool_name mom_attr_OPER_TITLE1 mom_attr_TOOL_TOOL_TITLE1

# retrieve attribute value

MOM_output_literal "( The value for the operation attribute is/
$mom_attr_OPER_TITLE1 )"

=====

# Add the following lines in custom command procedure called EXPRESSION
# The following lines allow the use of global variables in this procedure

global mom_path_name MOM_ask_ess_exp_value
global mom_sys_control_in mom_sys_control_out

# call in the control in & out codes

set ci " "
set co " "
if {[info exists mom_sys_control_in] == 1 } { set ci $mom_sys_control_in }
if {[info exists mom_sys_control_out] == 1 } { set co $mom_sys_control_out }

# retrieve expression value

set expl [MOM_ask_ess_exp_value fixture1];#fixture 1 is the expression name
MOM_set_seq_off
MOM_output_literal "$co THE VALUE FOR EXPRESSION FIXTURE1 IS/
[format "%2.f" $expl] $ci"
MOM_set_seq_on
=====

=====

# This custom command will grab the group program name and strip off the
# first Alpha character and dump the rest at the start of the output file. To use
# this custom command, change or create a program name
# with what you want i.e. P123456789, the custom command will then drop the
# "P" and output the rest

global mom_group_name
```

A

```

set partno_txt " NONE "
if {[info exists mom_group_name] == 1} { set partno_txt $mom_group_name}

MOM_set_seq_off
MOM_output_literal "PARTNO [string toupper [string range $partno_txt 1 end]]"
MOM_set_seq_on

=====

#
# proc MOM_end_of_path {} CALLED AT THE END OF EACH PATH
#
# add a custom command to the Start_of_program and in it
# add this line "set accumulated_time 0". This will start the time clock at zero
#
global mom_machine_time accumulated_time mom_operation_name
global mom_sys_control_in mom_sys_control_out
#
# call in the control in & out codes
#
set ci " "
set co " "
if {[info exists mom_sys_control_in] == 1 } { set ci $mom_sys_control_in }
if {[info exists mom_sys_control_out] == 1 } { set co $mom_sys_control_out }
set op_time [expr $mom_machine_time - $accumulated_time]
MOM_set_seq_off
MOM_output_literal "$co-----$ci"
MOM_output_literal "$co[string toupper [format "Operation: %s Time: "%.2f"/
  $mom_operation_name $op_time]]$ci"
MOM_output_literal "$co-----$ci"
MOM_set_seq_on
set accumulated_time $mom_machine_time

=====

=====

#
# Add to start of path procedure
#
global mom_path_name mom_part_material_description
global mom_tool_name mom_oper_method mom_tool_catalog_number
global mom_sys_control_in mom_sys_control_out mom_tool_name
global mom_tool_number
#
# call in the control in & out codes
#
set ci " "
set co " "
if {[info exists mom_sys_control_in] == 1 } { set ci $mom_sys_control_in }
if {[info exists mom_sys_control_out] == 1 } { set co $mom_sys_control_out }

set cat_info "NONE ENTERED"
if {[info exists mom_tool_catalog_number] == 1} {set cat_info/
  $mom_tool_catalog_number}

MOM_set_seq_off
MOM_output_literal "$co PATH NAME : [string toupper $mom_path_name]/
WITH TOOL NO: [string toupper $mom_tool_number] $ci"

```

```

MOM_output_literal "$co TOOL NAME : [string toupper $mom_tool_name]/
TOOL CATALOG NO. [string toupper $cat_info] $ci"
MOM_output_literal "$co METHOD : [string toupper $mom_oper_method]/
$ci"
MOM_set_seq_on

=====

=====

# custom command to reset the sequence number to match the tool number

global mom_sys_seqnum_start
global mom_sys_seqnum_incr
global mom_sys_seqnum_freq
global mom_tool_number

MOM_reset_sequence $mom_tool_number $mom_sys_seqnum_incr/ $mom_sys_seqnum_freq

=====

#
# add this custom command to the spindle rpm event under machine control #event

global mom_spindle_rpm

if {[info exists mom_spindle_rpm] == 1} {
if {$mom_spindle_rpm <= 450 } {
MOM_output_literal "M41"
} elseif { $mom_spindle_rpm > 450 && $mom_spindle_rpm <=900 } {
MOM_output_literal "M42"
} else { MOM_output_literal "M43"}
}

=====

#
# Reformat the time stamp to make it organized
#
set tmp [string range $mom_date 4 10][string range $mom_date 20 23]
MOM_output_literal "(DATE: $tmp)"
set tmp [string range $mom_date 11 15]
MOM_output_literal "(TIME: $tmp)"

=====

=====

global mom_tool_catalog_number mom_tool_number
global mom_sys_control_in mom_sys_control_out mom_tool_name

# call in the control in & out codes

set ci " "
set co " "
if {[info exists mom_sys_control_in] == 1 } { set ci $mom_sys_control_in }
if {[info exists mom_sys_control_out] == 1 } { set co $mom_sys_control_out }

set cat_info "NONE ENTERED"

```

A

```
if {[info exists mom_tool_catalog_number] == 1} {set cat_info $mom_tool_catalog_number}

MOM_set_seq_off
MOM_output_literal "$co----- $ci"
MOM_output_literal "$co TOOL NAME : [string toupper $mom_tool_name]/
TOOL CATALOG NO. [string toupper $cat_info] $ci"
MOM_output_literal "$co----- $ci"
MOM_set_seq_on

=====
```



## Appendix

# *B Advanced Post Building topics*

## B

### Purpose

This appendix describes procedures to build advanced, complex, custom **NX/Post** post processors, modify existing post processors and discusses, in detail, the various components which comprise **NX/Post**.

### Objective

Upon completion of this lesson, you will be able to:

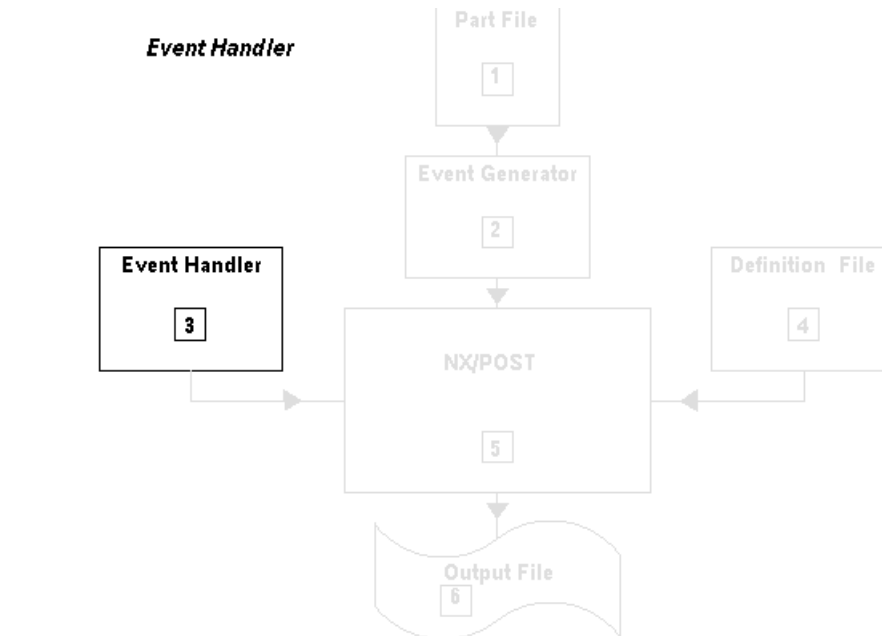
- Modify and customize **NX/Post** post processors.
- Construct and use Definition files and Event Handlers.
- Debug a **NX/Post** post processor.
- Post process from NX.
- Post process from the operating system.

**B**

## Guide to modifying and customizing existing post processors

The key to modifying and customizing existing post processors is the ability to edit and modify the **Event Handler** (.tcl file) and the **Definition file** (.def file).

### Event Handler



- 1 Part File
- 2 EVENT GENERATOR extracts information from the part and sends information to the postprocessor
- 3 EVENT HANDLER processes each EVENT according to instructions
- 4 EVENTS processed by the EVENT HANDLER are formatted according to the Definition File
- 5 NX/Post Post Processor controls entire sequence
- 6 Postprocessed machine control instructions are written to an Output file

The **Event Handler** is a set of instructions that are developed for particular machine tool/controller combinations. Each **Event Handler** must contain a set of instructions for each type of Event that **NX/Post** will process. These instructions will define how the tool path data is processed and how each Event is executed at the machine tool.

These instructions are defined by the Tcl scripting language. The Tcl scripting language acts as the "translator" for **NX/Post**.

For each Event that the Event Handler will process, an existing Tcl procedure must exist. The **Manufacturing Output Manager** (MOM) will invoke this procedure when an Event is executed by the Event Generator. For each sequential event generated from the MOM, the Event Handler file will process the Event in sequential order. If you do not want an Event Handler

to process a particular Event type, do not include the Tcl procedure for that Event in the Event Handler.

The Tcl procedure name for each Event created must be identical to the Event name invoked by the Event Generator. For example, the procedure name for a start of program Event would be **MOM\_start\_of\_program**, for a tool change Event would be **MOM\_tool\_change**.

The parameters associated with each Event are passed to the Event Handler as global variables.

B

## Activity — Modify an event handler

In this activity you will perform various modifications to the Event Handler, which you created previously.

**Step 1:** Modify the procedure `start_of_program` in your `***_test_post.tcl` file (an example follows the last step-action item).

- ☐ Double click on the file `***_test_post.tcl`.
- ☐ Modify the **start\_of\_program** to output `"%4711"` without a sequence number.
- ☐ Add a header line with no `"N"`.
- ☐ Output the following globals: **mom\_date**; **mom\_logname**; **mom\_part\_name**.
- ☐ Add a command to turn the **Sequence Number** on.
- ☐ Output the following on a separate line: `"G40 G80 G17"`

**Example:** Find the line containing `MOM_start_of_program`.

After the line that reads:

**"LIST\_FILE\_HEADER ; #list header in commentary listing"**

insert the following:

```
MOM_set_seq_off ;#turns off the sequence number
MOM_output_literal "%4711" ;#dumps directly to output file
global mom_date mom_logname mom_part_name
MOM_output_literal ""
MOM_output_literal "Part Name :[string toupper [file tail $mom_part_name]]"
MOM_output_literal "Created by :[string toupper $mom_logname]"
MOM_output_literal "Creation date :[string to upper $mom_date]"
MOM_set_seq_on ;#turns sequence numbers on
MOM_output_literal "G40 G80 G17" ;#dumps directly to output file
```

- ☐ **Save** the file.
- ☐ Verify the output by posting the operations listed under `"T12345-A"` from the `mill_test.prt` file.

**Step 2:** Modify the procedure `start_of_path` in your `***_test_post.tcl` file (an example follows).

- ☐ Modify the **start\_of\_path** to output:
- ☐ **mom\_operation\_type**
- ☐ **mom\_path\_name**

- ☐ **mom\_tool\_name**
- ☐ **mom\_tool\_type**
- ☐ **mom\_tool\_number**

Insert the following After the line that reads:

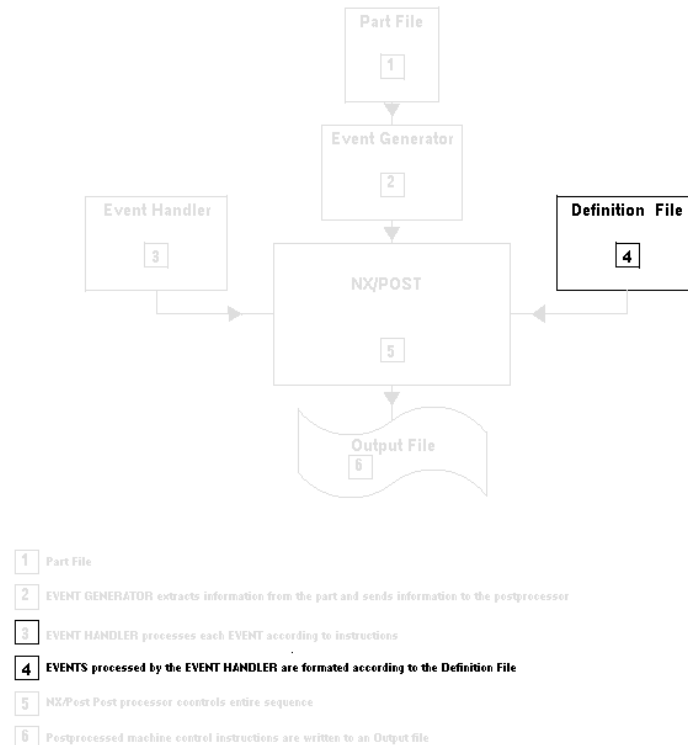
**"PB\_CMD\_start\_of\_operation\_force\_addresses"**

```
global mom_operation_type mom_path_name mom_tool_name
global mom_tool_type mom_tool_number
MOM_set_seq_off ;#turns off sequence numbers
MOM_output_literal "Operation Type":[string to upper $mom_operation_type]"
MOM_output_literal "Path Name :[string toupper $mom_part_name]"
MOM_output_literal "Tool Name :[string toupper $mom_tool_name]"
MOM_output_literal "Tool Type :[string to upper $mom_tool_type]"
MOM_output_literal "Tool No :[string to upper $mom_tool_number]"
MOM_set_seq_on ;#turns sequence numbers on
```

- ☐ **Save the file.**
  - ☐ Verify the output by posting the operations listed under the parent **"T12345-A"** from the **mill\_test.prt** file.
- Step 3:** Modify the **\*\*\*\_test\_post.tcl** file to output G11 for a linear move, G12 for CLW move and G13 for CCLW move.
- ☐ In the **\*\*\*\_test\_post.tcl** file, locate the line **set mom\_sys\_linear\_code "1"** and change the **1** to **11**.
  - ☐ In the same file, locate the line **set mom\_sys\_circle\_code(CLW) "2"** and change the **2** to **12**.
  - ☐ In the same file, locate the line **set mom\_sys\_circle\_code(CCLW) "3"** and change the **3** to **13**.
  - ☐ **Save the file.**
  - ☐ Verify the output by posting the operations listed under the parent **"T12345-A"** from the **mill\_test.prt** file.

This completes the activity.

## Definition file



B

The **Definition** file contains information about the specific machine tool that you are creating the post processor for.

Most NC/CNC machines use addresses to instruct a machine how to perform a function. As an example, the addresses X, Y, and Z are used to store the value of the X, Y, and Z coordinate of the end position of a linear move. Each command in an NC/CNC program changes the state of the machine by changing the state of the machine's addresses. The information in the Definition file is used by NX/Post to format commands to a form that is understood by the machine tool controller.

Information in the Definition file pertains to machine attributes, such as tool changer, spindle speeds, and travel limits; addresses supported by the machine, such as X, Y, and Z for linear motion, S for spindle and F for feed rate; the attributes for each machine address, such as format, minimum and maximum travel; and a set of block templates that describes how the addresses are combined to perform an action by the machine tool. For example, a linear move can be designated by G01 X [Xval], Y [Yval], and Z [Zval].

An example of a Definition file can be found in Appendix A.

## Activity — Modify a definition file

**Step 1:** Modifying your\*\*\*\_test\_post.def file to insert a space between addresses.

- ☐ Double click on the file \*\*\*\_test\_post.def.
- ☐ Modify the file to output NC code with **two spaces** between the address.
- ☐ Change the value for word separator.
- ☐ **Save** the file.
- ☐ Verify the output by posting the operation "T12345-A" from the **mill\_test.prt** file.
- ☐ Modify the file to output NC code without a space between the address.
- ☐ Change the value for the word separator.
- ☐ **Save** the file.
- ☐ Verify the output by posting the operations under the parent "T12345-A" from the **mill\_test.prt** file.

**Step 2:** Modifying your\*\*\*\_test\_post.def file to alter the sequence number.

- ☐ If necessary, open the file \*\*\*\_test\_post.def.
- ☐ Modify the file to have the sequence number start at 1 and increment by 1.
- ☐ **Save** the file.
- ☐ Verify the output by posting the operations under the parent "T12345-A" from the **mill\_test.prt** file.
- ☐ Modify the file to have the sequence number start at 5, increment by 10 and output every other block increment.
- ☐ **Save** the file.
- ☐ Verify the output by posting the operations under "T12345-A" from the **mill\_test.prt** file.

**Step 3:** Modifying your \*\*\*\_test\_post.def file to alter the characteristics of the sequence number word.



- ☐ If necessary, open the file **\*\*\*\_test\_post.def**.
- ☐ Modify the file to have the sequence number start at 10 and increment by 10 and have no leading zeros (refer to Appendix D for format structure).
- ☐ **Save** the file.
- ☐ Verify the output by posting the operations under the parent **"T12345-A"** from the **mill\_test.prt** file.
- ☐ Modify the file to have the sequence number start at 10, increment by 10 and output leading zeros (format for sequence number to be 4 characters).
- ☐ **Save** the file.
- ☐ Verify the output by posting the operations under the parent **"T12345-A"** from the **mill\_test.prt** file.

**Step 4:** Modifying your\*\*\*\_test\_post.def file to alter the characteristics of the co-ordinate words.

- ☐ Modify the file to have co-ordinate output to contain 5 whole digits, 3 fractional digits, no leading zeros, trailing zeros and no decimal point.
- ☐ **Save** the file.
- ☐ Verify the output by posting the operations under the parent **"T12345-A"** from the **mill\_test.prt** file.
- ☐ Modify the file to have co-ordinate output to contain 4 whole digits, 4 fractional digits, leading and trailing zeros and no decimal point.
- ☐ **Save** the file.
- ☐ Verify the output by posting the operations under the parent **"T12345-A"** from the **mill\_test.prt** file.

**Step 5:** Modifying your\*\*\*\_test\_post.def file to control word output.

- ☐ Modify the file to use the Y value for X and the X value for Y for linear moves.
- ☐ **Save** the file.

- ☐ Verify the output by posting the operations under the parent "T12345-A" from the **mill\_test.prt** file.
- ☐ Add the Z co-ordinate to the first move.
- ☐ **Save** the file.
- ☐ Verify the output by posting the operations under the parent "T12345-A" from the **mill\_test.prt** file.

**Step 6:** Modify the output for the Event Cycle Drill.

- ☐ Output the following for the Event type Cycle Drill:  
G81 X Y E R  
X Y is the start position of the drill cycle.  
E is the depth of the hole (normally the Z value).  
R is the rapid position.
- ☐ **Save** the file.
- ☐ Verify the output by posting a cycle operation under the parent "T12345-A" from the **mill\_test.prt** file.

**Step 7:** Modifying your\*\*\*\_test\_post.def file to change the content of arc motion blocks.

- ☐ Modify the file to output the circle center co-ordinates for all arc motion blocks.
- ☐ **Save** the file.
- ☐ Verify the output by posting the operations under the parent "T12345-A" from the **mill\_test.prt** file.

**Step 8:** Alter the rapid move procedure.

- ☐ In place of G00, output G01 with an R.  
G01 X Y Z R
- ☐ **Save** the file.
- ☐ Verify the output by posting the operations under the parent "T12345-A" from the **mill\_test.prt** file.

**Step 9:** Alter the rapid move procedure.

- ☐ For all rapid moves, output F0.

G00 X Y Z F0

- ☐ **Save** the file.

- ☐ Verify the output by posting the operations under the parent "T12345-A" from the **mill\_test.prt** file.

This completes the activity.

**B**

## Machine Kinematics

Machine Kinematics apply to machine tool motions. NX/Post uses kinematic variables which define machine tool specific information used by the Event Generator. Kinematic variables defined by the Event Handler determine how the Event Generator converts the x, y, z, i, j, k tool path coordinates into machine tool coordinates. Kinematic variables are also used to define the basic type of machine such as lathe, mill or wire EDM.

Basic types of machines are defined by the kinematic variable **mom\_kin\_machine\_type**. Current values for this variable are:

- 3\_axis\_mill
- 4\_axis\_head
- 4\_axis\_table
- 5\_axis\_dual\_table
- 5\_axis\_dual\_head
- 5\_axis\_head\_table
- 2\_axis\_wedm
- 4\_axis\_wedm

When the machine type is set to lathe, all coordinates in the tool path are mapped to lathe coordinates. X, Y, Z tool path data is mapped to X, Z machine tool data.

When the machine type is set to mill, the x,y,z,i,j,k tool path coordinates are mapped to x, y, z, 4th-axis rotary and 5th-axis rotary coordinates. X, Y, Z, I, J, K tool path data is mapped to X,Y,Z, A, B machine tool data.

The machine tool type defines the number of axes of motion. Each rotary axis has the following characteristics:

- a plane of rotation
- travel limits
- direction of rotation
- zero position

## Circles

The kinematic variable, **mom\_kin\_arc\_output\_mode** determines the data and number of blocks that will be generated for arcs from the tool path. Associated with this variable is a parameter which will control arc output type. The parameter used can be one of the following:

- **Full\_Circle** - a **circle\_move** Event will be generated in 360 degree increments.
- **Quadrant** - a **circle\_move** Event will be generated for each quadrant boundary crossed.
- **linear arcs** - will cause the Event Generator to break the arcs into linear GOTO events.

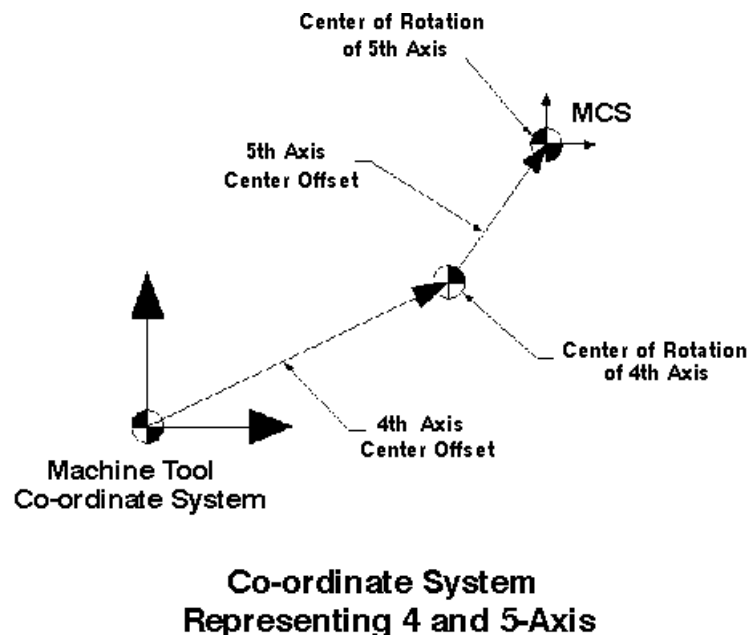
The kinematic variable, **mom\_kin\_arc\_valid\_plane**, determines what plane that the Event Generator will use to generate **circle\_move** events. The parameter used can be any of the following:

- **XYZ** - arcs will be generated in all three principal planes
- **XY** - arcs will be generated only in the XY plane
- **YZ** - arcs will be generated only in the YZ plane
- **ZX** - arcs will be generated only in the ZX plane

## Advanced Kinematics

For a 5-axis machine, one of the following values is assigned to the kinematics variable **mom\_kin\_machine\_type** to designate the machine type:

- **5\_axis\_dual\_table**
- **5\_axis\_head\_table**
- **5\_axis\_dual\_head**

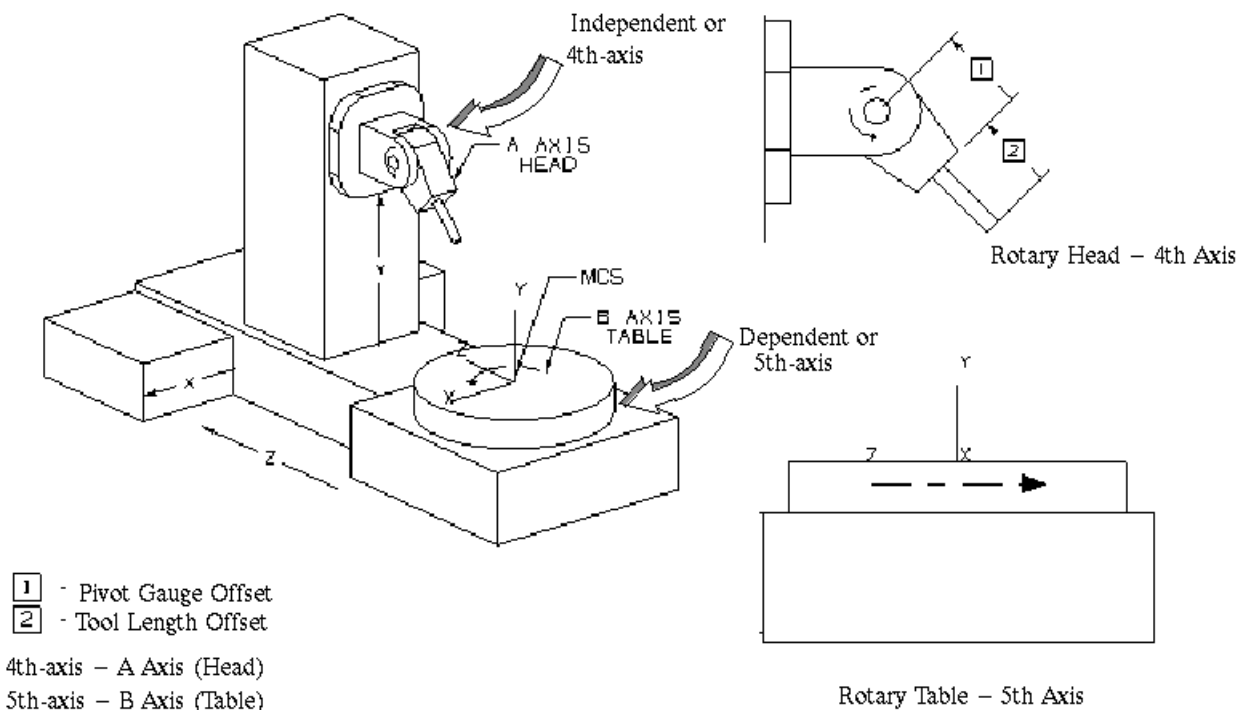


## Rotary axes designations

The rotary axis is the **independent** axis that does **not change** the orientation or plane of rotation when the second rotary axis moves. In a **5\_axis\_head\_table** configuration, the swivel head is always the independent or 4th-axis.

The rotary axis is the **dependent** or 5th-axis that **does change** the orientation or plane of rotation when the second rotary axis moves.

The C-Axis should never be set as the 4th axis.



<u>VARIABLE</u>	<u>VALUE</u>
mom_kin_5th_axis_center_offset(0)	0
mom_kin_5th_axis_center_offset(1)	0
mom_kin_5th_axis_center_offset(2)	0

## 5-Axis head table type

## 4th axis center offsets

The variable array **mom\_kin\_4th\_axis\_center\_offset** contains the offset values of the machine tool's zero position to the center of the 4th-axis table. In order for the post to correctly map the part co-ordinate system to the machine tool co-ordinate system, the offset value **must** be correct. The following parameters are used to designate these values:

**mom\_kin\_4th\_axis\_center\_offset(0)** X distance  
**mom\_kin\_4th\_axis\_center\_offset(1)** Y distance  
**mom\_kin\_4th\_axis\_center\_offset(2)** Z distance

**B**



## 5th axis center offsets

The variable array **mom\_kin\_5th\_axis\_center\_offset** (the distance from the center of the 5th-axis to the center of the 4th-axis), contains offset values and indicates that the 5th-axis of rotation does not intersect the 4th-axis of rotation. Calculation of the vector is dependent on both axes being at zero degrees. The value can be one or two co-ordinates, one is normally used. The perpendicular vector is measured in the plane of the 4th-axis from the axis of rotation of the 4th-axis to the axis of rotation of the 5th-axis. The following parameters are used to designate these values:

**mom\_kin\_5th\_axis\_center\_offset(0)** X distance  
**mom\_kin\_5th\_axis\_center\_offset(1)** Y distance  
**mom\_kin\_5th\_axis\_center\_offset(2)** Z distance

**B**

Axis rotation (standard or reverse)

The axis rotation parameter, **mom\_kin\_4th (5th)\_axis\_rotation**, controls how UG/POST translates the i,j,k tool axis information into rotary motion. The following parameters are used to designate axis rotation:

<b>mom_kin_4th_axis_rotation</b>	<b>Standard</b>
<b>mom_kin_4th_axis_rotation</b>	<b>Reverse</b>
<b>mom_kin_5th_axis_rotation</b>	<b>Standard</b>
<b>mom_kin_5th_axis_rotation</b>	<b>Reverse</b>

B

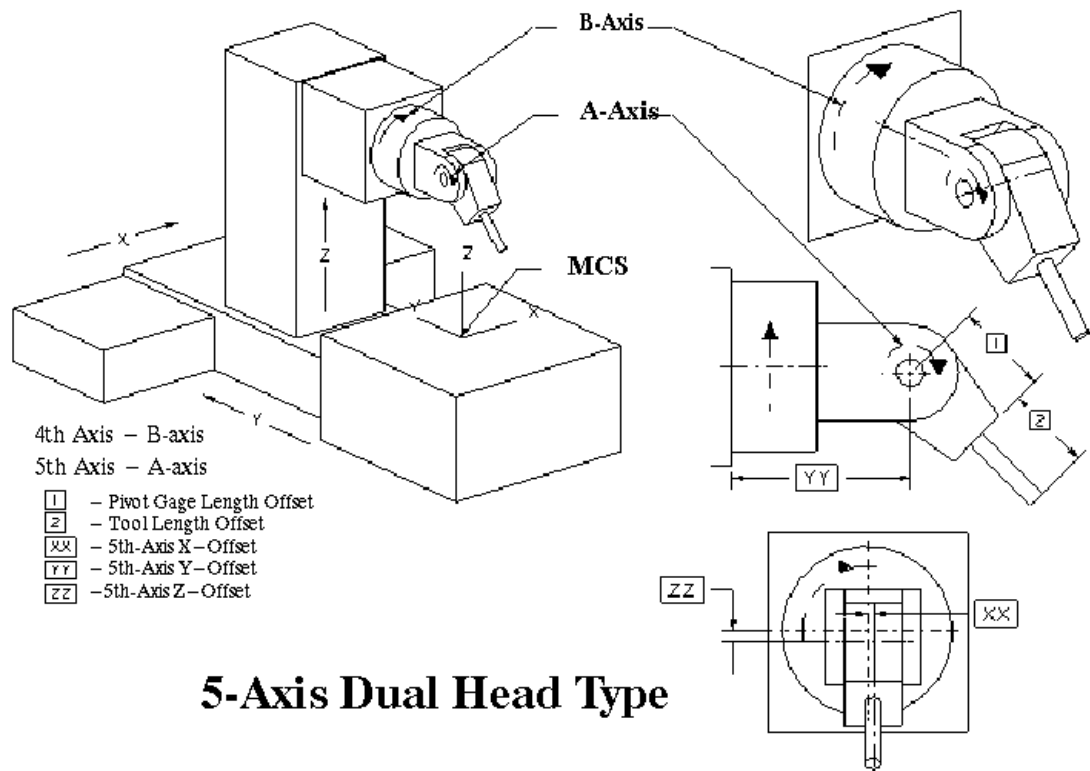
The following guidelines are used to determine type of rotation:

- Most CNC machines are **Standard**, with rotation or swiveling in a clockwise direction to a larger angle (when in question use the **Standard** convention, verify that the output is correct).
- In the XY plane looking down the Z axis from positive to negative, rotation clockwise to a larger angle is **Standard**.
- In the ZX plane looking down the Y axis from positive to negative, rotation clockwise to a larger angle is **Standard**.
- In the YZ plane looking down the X axis from negative to position, rotation clockwise to a larger angle is **Standard**.

B

Each of the mentioned kinematic variables, involving rotation may be set to **standard** or **reverse** as required by the CNC controller.

For example, a dual head machine, as show in the following diagram, can orient the tool along a (1,0,0) vector by rotation of the A-axis and B-axis by 90 degrees. If the axis rotation of the A axis is set to **reverse**, then the A axis angle will be -90 degrees for the same tool axis vector.



The following defines the tool kinematics for 5-Axis Dual Head type machine tools.

set mom_kin_machine_type	5_axis_dual_head
set mom_kin_4th_axis_plane	ZX
set mom_kin_4th_axis_center_offset(0)	0.0
set mom_kin_4th_axis_center_offset(1)	0.0
set mom_kin_4th_axis_center_offset(2)	0.0
set mom_kin_4th_axis_rotation	standard
set mom_kin_4th_axis_zero_position	0.0
set mom_kin_pivot_gauge_offset	100.00
set mom_kin_5th_axis_plane	YZ
set mom_kin_5th_axis_center_offset(0)	XX (5th axis x-offset)
set mom_kin_5th_axis_center_offset(1)	YY (5th axis y-offset)
set mom_kin_5th_axis_center_offset(2)	ZZ (5th axis z-offset)
set mom_kin_5th_axis_rotation	standard
set mom_kin_5th_axis_zero_position	0.0

The distance from the machine tool zero to the center of the 4th-axis is not needed in this type of machine and **must be** set to **zero**.

## Zero position offset

When both rotary angles on a machine are set to zero, the tool will be aligned to the Z-axis of the machine tool co-ordinate system.

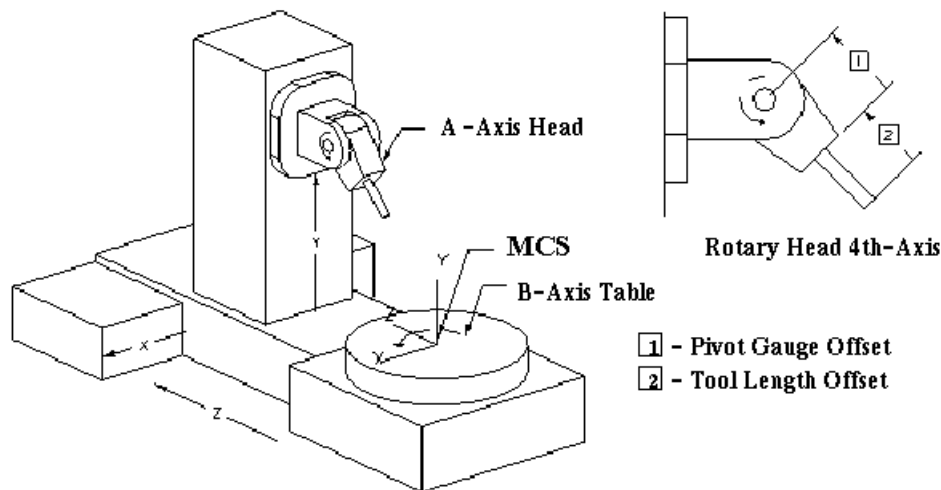
The zero position of a rotary axis on some machine tools (typically dual head and head table machines) is set at a finite angular value.

This value may be defined as a parameter in the kinematic variable **mom\_kin\_4th\_axis\_zero\_position** and **mom\_kin\_5th\_axis\_zero\_position** for the 4th and 5th-axis respectively.

Referring to the following diagram, which represents a head\_table 5-axis machine, the zero position for the head may be set to -90 degrees. When the head is at 0 degrees, the tool will be aligned with the Y-axis of the machine tool co-ordinate system, towards the table. The kinematic variables to perform this function would be set as follows:

set <b>mom_kin_4th_axis_zero_position</b>	<b>-090.0</b>
set <b>mom_kin_5th_axis_zero_position</b>	<b>0.0</b>

If the tool axis vector were (0,0,1) the output for the A-axis rotary position would be 90.0 since the zero position is at -90 degrees.



### 5-Axis Head Table Type

The following defines the tool kinematics for 5-Axis Head Table type machine tools.

set mom_kin_machine_type	5_axis_head_table
set mom_kin_4th_axis_plane	YZ
set mom_kin_4th_axis_center_offset(0)	0.0
set mom_kin_4th_axis_center_offset(1)	0.0
set mom_kin_4th_axis_center_offset(2)	0.0
set mom_kin_4th_axis_rotation	standard
set mom_kin_4th_axis_zero_position	0.0
set mom_kin_pivot_gauge_offset	100.0
set mom_kin_5th_axis_plane	ZX
set mom_kin_5th_axis_center_offset(0)	XX
set mom_kin_5th_axis_center_offset(1)	YY
set mom_kin_5th_axis_center_offset(2)	ZZ
set mom_kin_5th_axis_rotation	standard
set mom_kin_5th_axis_zero_position	0.0

## Pivot point

The pivot point is defined as the point about which the rotary head tilts. On a dual head machine this point is defined as the 4th and 5th-axis head.

For standard 5-axis head table and dual head machines this point may be defined by the following kinematic variable:

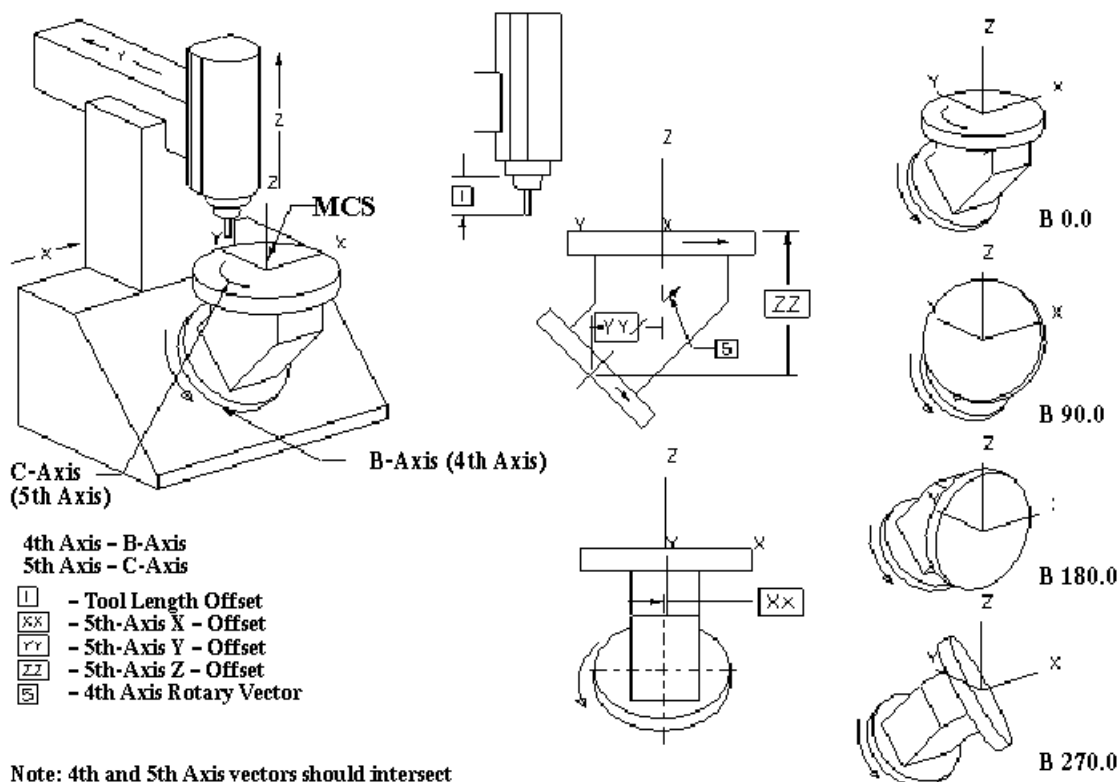
- **mom\_kin\_pivot\_gauge\_offsetvalue**

This offset is **measured** from the **spindle gauge point** to the **point of rotation of the head**.

For the machine type shown in the following diagram, the pivot point is defined by the following kinematic array variable:

<b>mom_kin_gauge_to_pivot</b>	<b>(0)XX (X-Head Offset)</b>
<b>mom_kin_gauge_to_pivot</b>	<b>(1)XX (Y-Head Offset)</b>
<b>mom_kin_gauge_to_pivot</b>	<b>(2)XX (Z-Head Offset)</b>

On these machines, the pivot point lies on the 4th axis rotation vector.



## 5-Axis Dual Table Special Case (Deckel Maho DMUxxV)

For machine tools with axes which are not in principal planes you must install the advanced kinematics library in the auxiliary directory and use

Tcl procedures to call this library. This library is an optional item. Use the **NONE** value for the axis plane and direction cosines or angles to specify the axis of rotation of the table.

set mom_kin_machine_type	5_axis_head_table
set mom_kin_4th_axis_plane	NONE
set mom_kin_4th_axis_center_offset(0)	0.0
set mom_kin_4th_axis_center_offset(1)	0.0
set mom_kin_4th_axis_center_offset(2)	0.0
set mom_kin_4th_axis_rotation	standard
set mom_kin_4th_axis_zero_position	0.0

Using direction cosines

set mom_kin_4th_axis_vector(0)	0.0000
set mom_kin_4th_axis_vector(1)	1.0000
set mom_kin_4th_axis_vector(2)	1.0000

Using two angles

set mom_kin_4th_axis_angles(0)	90.0
set mom_kin_4th_axis_angles(1)	45.0

The distance from the spindle gauge point to the pivot point (the point about which the head rotates) is defined using the kinematics array variable **mom\_kin\_gauge\_to\_pivot**.

set mom_kin_gauge_to_pivot(0)	XX (Head Offset X direction)
set mom_kin_gauge_to_pivot(1)	YY (Head Offset Y direction)
set mom_kin_gauge_to_pivot(2)	ZZ (Head Offset Z direction)
set mom_kin_5th_axis_plane	NONE
set mom_kin_5th_axis_center_offset(0)	0
set mom_kin_5th_axis_center_offset(1)	0.0
set mom_kin_5th_axis_center_offset(2)	0.0
set mom_kin_5th_axis_rotation	standard
set mom_kin_5th_axis_zero_position	0.0

Using direction cosines :

set mom_kin_5th_axis_vector(0)	0.0000
set mom_kin_5th_axis_vector(1)	0.0000
set mom_kin_5th_axis_vector(2)	1.0000

Using two angles:

set mom\_kin\_5th\_axis\_angles(0)0.0 set mom\_kin\_5th\_axis\_angles(1)0.0

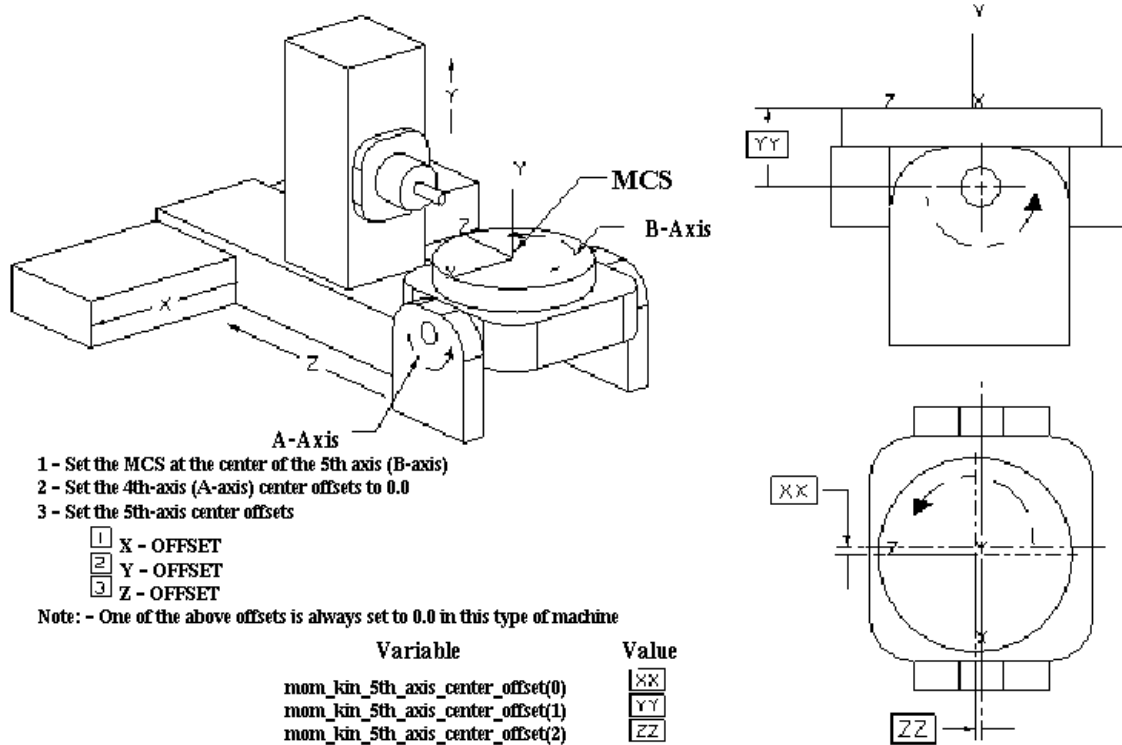


## Dual table dynamics

In the figure below, the B-Axis table is mounted on the A-Axis table. The MCS is set to the center of the 5th-axis table when creating operations.

If the machine tool zero is not at the center of the B-Axis table (Independent or 5th-axis), then the distance from the machine tool zero to the table center must be entered in the 4th-axis center offsets kinematic variable.

The distance from the 4th to 5th-axis must be set in 5th-axis center offsets kinematic variable **mom\_kin\_5th\_axis\_center\_offset**.



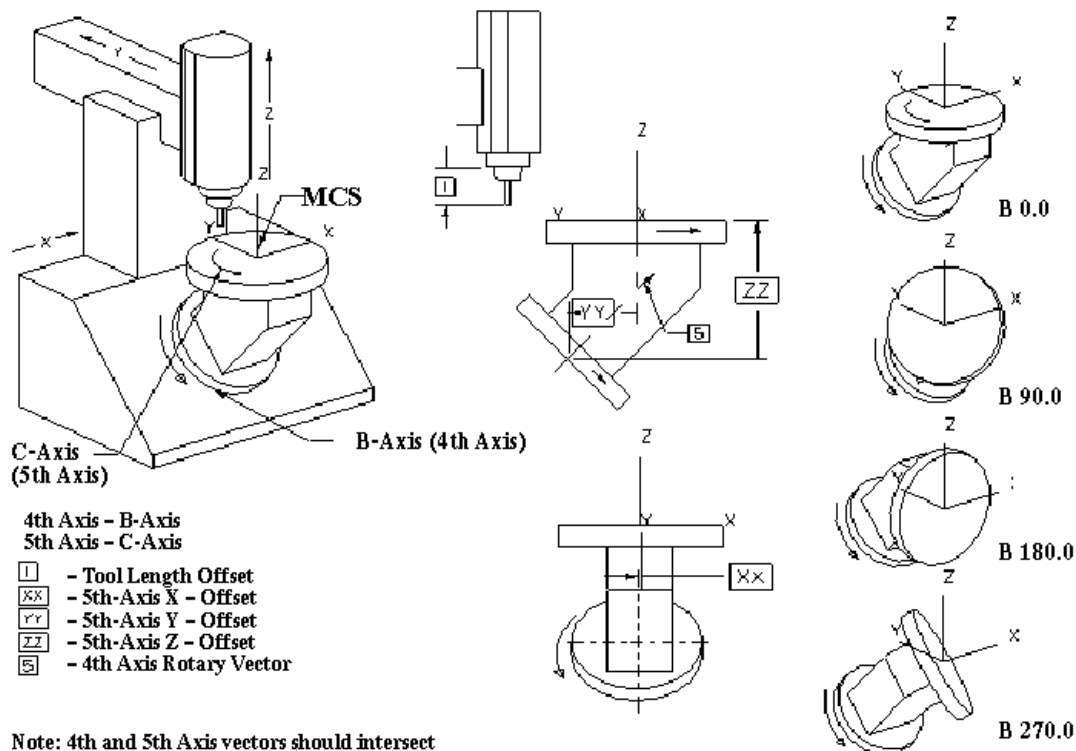
## Dual Table Type 5-Axis

The following defines the tool kinematics for 5-Axis Dual Table type machine tools:

set mom_kin_machine_type	5_axis_dual_table
set mom_kin_4th_axis_plane	YZ
set mom_kin_4th_axis_center_offset(0)	0.0
set mom_kin_4th_axis_center_offset(1)	0.0
set mom_kin_4th_axis_center_offset(2)	0.0
set mom_kin_4th_axis_rotation	standard
set mom_kin_4th_axis_zero_position	0.0
set mom_kin_5th_axis_plane	ZX
set mom_kin_5th_axis_center_offset(0)	XX
set mom_kin_5th_axis_center_offset(1)	YY
set mom_kin_5th_axis_center_offset(2)	ZZ
set mom_kin_5th_axis_rotation	standard
set mom_kin_5th_axis_zero_position	0.0

## Special case 5-axis dual table

For machine tools with their axes not in principal planes, you must use the NONE value for the axis plane and either direction cosines or angles to specify the axis of rotation of the table.



### 5-Axis Dual Table Special Case (Deckel Maho DMUxxV)

The following defines variables for the machine type and 4th-axis for the machine tool shown above:

set mom_kin_machine_type	5_axis_dual_table
set mom_kin_4th_axis_plane	NONE
set mom_kin_4th_axis_center_offset(0)	0.0
set mom_kin_4th_axis_center_offset(1)	0.0
set mom_kin_4th_axis_center_offset(2)	0.0
set mom_kin_4th_axis_rotation	standard
set mom_kin_4th_axis_zero_position	0.0

Using direction cosines:

set mom_kin_4th_axis_vector(0)	0.0000
set mom_kin_4th_axis_vector(1)	-1.0000
set mom_kin_4th_axis_vector(2)	1.0000

Using two angles:

```
set mom_kin_4th_axis_angles(0)      270.0
set mom_kin_4th_axis_angles(1)      45.0
```

The distance from the center of the C-axis table to the point about which the 4th-axis or independent axis rotates, is defined using the kinematics array variable **mom\_kin\_5th\_axis\_center\_offset**. The 4th and 5th-axis rotation vector are usually co-planar.

The following defines variables for the 5th-axis for the previously shown machine tool:

```
set mom_kin_5th_axis_plane           NONE
set mom_kin_5th_axis_center_offset(0) 1.0
set mom_kin_5th_axis_center_offset(1) -100.0
set mom_kin_5th_axis_center_offset(2) -75.0
set mom_kin_5th_axis_rotation         standard
set mom_kin_5th_axis_zero_position    0.0
```

Using direction cosines:

```
set mom_kin_5th_axis_vector(0)        0.0000
set mom_kin_5th_axis_vector(1)        0.0000
set mom_kin_5th_axis_vector(2)        1.0000
```

Using two angles:

```
set mom_kin_5th_axis_angles(0)        0.0
set mom_kin_5th_axis_angles(1)        0.0
```

## UG/Post Postprocessing using Runugpost

Runugpost is an interactive menu utility which will allow you to execute the UG/Post postprocessor **outside** of aNX session.

The utility file name and executable command follows:

System	Filename	Execute Command
Windows	ugpost	ugpost
Unix	ugpost	ugpost

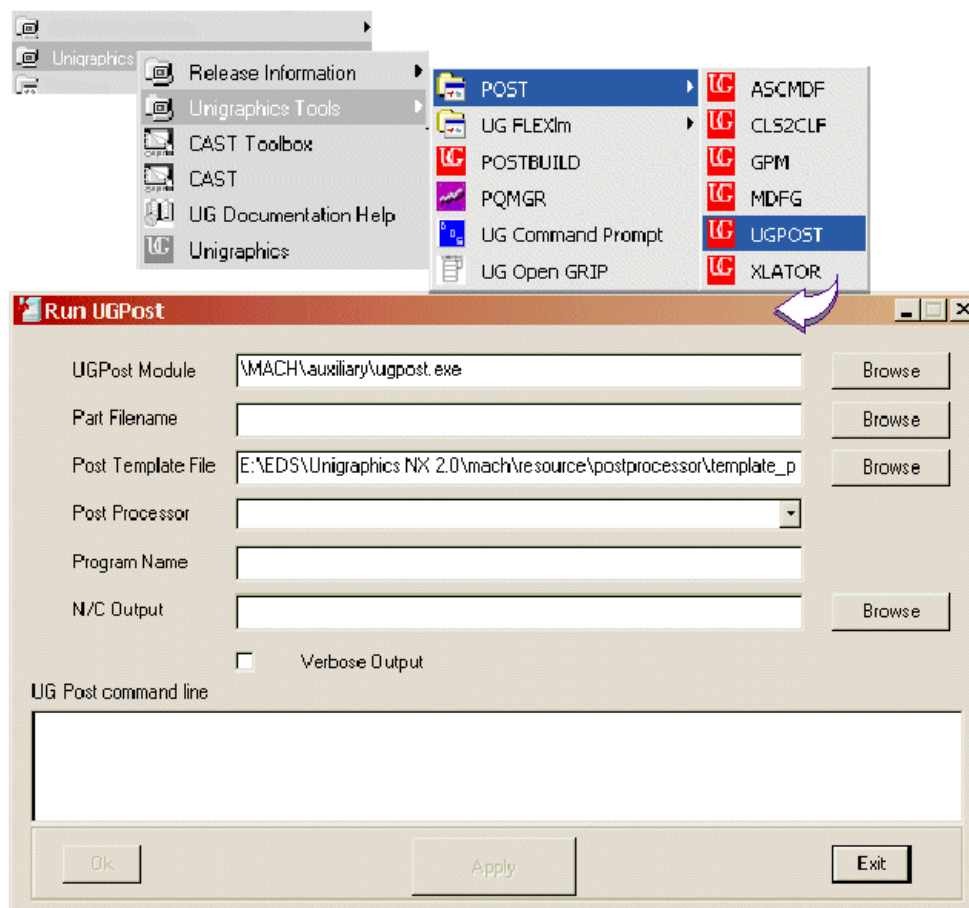
SystemFilenameExecute Command

Windows ugpostugpost

Unixugpostugpost

You can also activate Runugpost from the Windows desktop by selecting

**START ® Programs ® NX ® Post Tools ® ugpost**



## Summary

The flexibility of Post Builder allows for the creation of postprocessors for most milling, turning and wire edm applications. In those situations that the Post Builder cannot provide the output that is required, modification of the Definition and Event files are required. This appendix covers topics in:

- The use and modification of Definition files.
- The use and modification of Event Handlers.
- Advanced techniques to customize and modify UG/Post postprocessor.
- Postprocessing from the operating system.

# Index

## B

Best Practices	11-3
Definition file	11-3
Overview	11-1
block	2-41
block component	2-31
editing of	2-33
Block word address symbols	2-42

## C

Component Window	2-28
Custom Commands	8-2
Customizing existing postprocessors	
Customizing existing Post Processors	B-3

## D

Definition file	2-3, B-7
-----------------	----------

## E

Event Generator	1-3
Event Handler	1-3, 2-3, B-3

## F

Files Preview tab	2-62
format	2-42

## I

Integration, Simulation and Verification	
machine tool driver	10-3
overview	10-2

## L

Listing File & Output Control tab	2-56
-----------------------------------	------

## M

Machine Kinematics	B-12
Machine Tool tab	2-16
Default button	2-16
Restore button	2-16
Manufacturing Output Manager	1-4
Mill-turn	
creating using Post Builder	6-3, 6-12
types of	6-2, 6-10
Modifying existing postprocessors	B-3

## N

N/C Data Definitions tab	2-41
--------------------------	------

## O

Other data elements	2-42
Output File	1-3

## P

Parameter Window	2-28
Postprocessing	
definition of	1-2
using Runugpost	B-29
Postprocessor	
UG/Post Execute	1-2
Program and Tool Path tab	2-19
Custom Commands	2-27
G Codes	2-19
M Codes	2-19
Program	2-19
Word Sequencing	2-27
Word summary	2-20
Data type	2-20
Decimals	2-20
Fractions	2-20
Integers	2-20

- Leader/Code . . . . . 2-20
- Leading zeros . . . . . 2-20
- Plus (+) . . . . . 2-20
- Trailing Zeros . . . . . 2-20
- Word . . . . . 2-20
- R**
- Runugpost . . . . . B-29
- S**
- Sequence . . . . . 2-29
  - Operation End . . . . . 2-31
  - Operation Start . . . . . 2-30
  - Program End . . . . . 2-31
  - Program Start . . . . . 2-30
  - Tool Path Events . . . . . 2-30
    - Canned Cycle Events . . . . . 2-31
    - Machine Control Events . . . . . 2-31
    - Motion Events . . . . . 2-31
- T**
- Tcl . . . . . B-3
- Tcl Procedures . . . . . 8-3
- template\_post.dat file . . . . . 2-10
- Trash bin . . . . . 2-32
- U**
- UG/Post
  - User Defined Events . . . . . 9-2
- UG/Post Builder . . . . . 1-4
  - 5-axis . . . . . 4-2
  - 5-axis parameters . . . . . 4-6
  - Balloon Tip . . . . . 2-7
  - Context Help . . . . . 2-7
  - Creating a new post . . . . . 2-9
  - Cue Line . . . . . 2-6
  - Help Tool Bar . . . . . 2-6
  - lathe . . . . . 5-2
    - parameters . . . . . 5-3
  - Menu Bar . . . . . 2-6
  - Overview . . . . . 2-3
  - Parameter file . . . . . 2-3
  - Parameters . . . . . 2-15
  - Tool Bar . . . . . 2-6
  - User's Manual
    - Users Manual . . . . . 2-7
- UG/Post Builder releases . . . . . 11-3
- W**
- Wire EDM
  - 2-axis . . . . . 3-2
  - 4-axis . . . . . 3-2
  - controller type . . . . . 3-2
    - generic . . . . . 3-3
    - library . . . . . 3-3
    - user's . . . . . 3-3
- word . . . . . 2-41





Education Services

UGS Education Services offers a blend of training solutions for all of our product lifecycle management products.

Our Online Store “Learning Advantage” was developed to provide our customers with “just in time” training for the latest in application developments.

Here are some of the Learning Advantages:

- Customers have direct access
- Self-paced course layout
- Online Assessments
- Just in time training for the latest release

To learn more about the “Learning Advantage” visit our website <http://training.ugs.com> or email us at training @ugs.com

L  
E  
A  
R  
N  
I  
N  
G  
  
A  
D  
V  
A  
N  
T  
A  
G  
E

Training solutions for all of our PLM software:



training@ugs.com

<http://training.ugs.com>

This page left blank intentionally.



## STUDENT PROFILE

In order to stay in tune with our customers we ask for some background information. This information will be kept confidential and will not be shared with anyone outside of Education Services.

Please “Print”...

Your Name \_\_\_\_\_ U.S. citizen ☐ Yes ☐ No

Course Title/Dates \_\_\_\_\_ / \_\_\_\_\_ thru \_\_\_\_\_

Hotel/motel you are staying at during your training \_\_\_\_\_

Planned departure time on last day of class \_\_\_\_\_

Employer \_\_\_\_\_ Location \_\_\_\_\_

Your title and job responsibilities \_\_\_\_\_ / \_\_\_\_\_

Industry: ☐ Auto ☐ Aero ☐ Consumer products ☐ Machining ☐ Tooling ☐ Medical ☐ Other

Types of products/parts/data that you work with \_\_\_\_\_

Reason for training \_\_\_\_\_

Please verify/add to this list of training for *Unigraphics, I-deas, Imageware, Teamcenter Mfg., Teamcenter Eng. (I-Man), Teamcenter Enterprise (Metaphase), or Dimensional Mgmt./Visualization*. **Medium** means Instructor-lead (**IL**), On-line (**OL**), or Self-paced (**SP**)

Software	From Whom	When	Course Name	Medium

Other CAD/CAM/CAE /PDM software you have used \_\_\_\_\_

Please “check”! your ability/knowledge in the following...

<u>Subject</u>	<u>None</u>	<u>Novice</u>	<u>Intermediate</u>	<u>Advanced</u>
CAD modeling	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CAD assemblies	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CAD drafting	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CAM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CAE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
PDM – data management	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
PDM – system management	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Platform (operating system) \_\_\_\_\_

*Thank you for your participation and we hope your training experience will be an outstanding one.*

This page left blank intentionally.

# Post Building Techniques V5.0 Course Agenda

## Day 1 Morning

- Introduction & Overview
- Lesson 1. The UG Post Processor
- Workbook Section 1 Project Description
- Lesson 2. Post Builder
- Workbook Section 2 Collecting Data for Post Processor Creation

### Afternoon

- Lesson 2. Post Builder
- Workbook Section 2 Collecting Data for Post Processor Creation
- Workbook Section 3 Building a Post Processor with Post Builder

---

## Day 2 Morning

- Lesson 2 Post Builder
- *Workbook Section 3* *Building a Post Processor with Post Builder*
- Lesson 3. Wire EDM

### Afternoon

- Lesson 4. Post Builder for 5-Axis Mill Applications
- *Workbook Section 4* *Program and Tool Path Functions*
- Lesson 5. Post Builder for Lathe Applications

---

## Day 3 Morning

- Lesson 6. Creating Mill Turn Post Processors
- Lesson 7. Tcl Basics for Post Builder
- *Workbook Section 5* *Post Builder Data Definitions*
- 

### Afternoon

- Lesson 8. Customizing a Post Processor
- *Workbook Section 6* *Post Builder Output Settings*
- *Workbook Section 7* *Post Processor Customization*

---

## Day 4 Morning

- Lesson 8. Customizing a Post Processor
- Lesson 9. User Defined Events

### Afternoon

- Lesson 10. Virtual NC Controller
  - Lesson 11. Best Practices
-



# Accelerators

The following Accelerators can be listed from within an NX session by choosing Information→Custom Menubar→Accelerators.

<b>Function</b>	<b>Accelerator</b>
File→New...	Ctrl+N
File→Open...	Ctrl+O
File→Save	Ctrl+S
File→Save As...	Ctrl+Shift+A
File→Plot...	Ctrl+P
File→Execute→Grip...	Ctrl+G
File→Execute→Debug Grip...	Ctrl+Shift+G
File→Execute→NX Open...	Ctrl+U
Edit→Undo	Ctrl+Z
Edit→Cut	Ctrl+X
Edit→Copy	Ctrl+C
Edit→Paste	Ctrl+V
Edit→Delete...	Ctrl+D or Delete
Edit→Selection→Top Selection Priority - Feature	F
Edit→Selection→Top Selection Priority - Face	G
Edit→Selection→Top Selection Priority - Body	B
Edit→Selection→Top Selection Priority - Edge	E
Edit→Selection→Top Selection Priority - Component	C
Edit→Selection→Select All	Ctrl+A
Edit→Show and Hide→Show and Hide... (by type)	Ctrl+W
Edit→Show and Hide→Hide...	Ctrl+B
Edit→Show and Hide→Invert Shown and Hidden	Ctrl+Shift+B
Edit→Show and Hide→Show...	Ctrl+Shift+K
Edit→Show and Hide→Show All	Ctrl+Shift+U
Edit→Transform...	Ctrl+T
Edit→Object Display...	Ctrl+J
View→Operation→Zoom...	Ctrl+Shift+Z
View→Operation→Rotate...	Ctrl+R
View→Operation→Section...	Ctrl+H
View→Layout→New...	Ctrl+Shift+N
View→Layout→Open...	Ctrl+Shift+O
View→Layout→Fit All Views (only with multiple views)	Ctrl+Shift+F
View→Layout→Fit	Ctrl+F
View→Visualization→High Quality Image...	Ctrl+Shift+H
View→Information Window	F4
Hide or show the current dialog box	F3
View→Reset Orientation	Ctrl+F8
Insert→Sketch...	S
Insert→Design Feature→Extrude...	X
Insert→Design Feature→Revolve...	R
Insert→Trim→Trimmed Sheet...	T
Insert→Sweep→Variational Sweep...	V

Format→Layer Settings...	Ctrl+L
Format→Visible in View...	Ctrl+Shift+V
Format→WCS→Display	W
Tools→Expression...	Ctrl+E
Tools→Journal→Play...	Alt+F8
Tools→Journal→Edit	Alt+F11
Tools→Macro→Start Record...	Ctrl+Shift+R
Tools→Macro→Playback...	Ctrl+Shift+P
Tools→Macro→Step...	Ctrl+Shift+S
Information→Object...	Ctrl+I
Analysis→Curve→Refresh Curvature Graphs	Ctrl+Shift+C
Preferences→Object...	Ctrl+Shift+J
Preferences→Selection...	Ctrl+Shift+T
Start→Modeling...	M or Ctrl+M
Start→All Applications→Shape Studio...	Ctrl+Alt+S
Start→Drafting...	Ctrl+Shift+D
Start→Manufacturing...	Ctrl+Alt+M
Start→NX Sheet Metal...	Ctrl+Alt+N
Start→Assemblies	A
Help→On Context...	F1
Refresh	F5
Fit	Ctrl+F
Zoom	F6
Rotate	F7
Orient View-Trimetric	Home
Orient View-Isometric	End
Orient View-Top	Ctrl+Alt+T
Orient View-Front	Ctrl+Alt+F
Orient View-Right	Ctrl+Alt+R
Orient View-Left	Ctrl+Alt+L
Snap View	F8



Evaluation – Delivery  
Post Building Techniques V5.0, Course #MT11060  
 Dates \_\_\_\_\_ thru \_\_\_\_\_

Please share your opinion in all of the following sections with a “check” in the appropriate box:

**Instructor:** \_\_\_\_\_ ☒

If there were 2 instructors, please evaluate the 2nd instructor with “X’s”

**Instructor:** \_\_\_\_\_ ☒

	STRONGLY DISAGREE	DISAGREE	SOMEWHAT DISAGREE	SOMEWHAT AGREE	AGREE	STRONGLY AGREE
1. ...clearly explained the course objectives.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2. ...was knowledgeable about the subject.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3. ...answered my questions appropriately.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4. ... encouraged questions in class.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5. ...was well spoken and a good communicator.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6. ...was well prepared to deliver the course.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7. ...made good use of the training time.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
8. ...conducted themselves professionally.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
9. ...used examples relevant to the course and audience.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
10. ...provided enough time to complete the exercises.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
11. ...used review and summary to emphasize important information.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
12. ...did all they could to help the class meet the course objectives.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

**Comments on overall impression of instructor(s):**

Overall impression of instructor(s)..... Poor ☐ ☐ ☐ ☐ ☐ ☐ Excellent

Suggestions for improvement of course delivery: \_\_\_\_\_

What you liked best about the course delivery: \_\_\_\_\_

**Class Logistics:**

1. The training facilities were comfortable, clean, and provided a good learning environment.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2. The computer equipment was reliable.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3. The software performed properly.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4. The overhead projection unit was clear and working properly.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5. The registration and confirmation process was efficient.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

**Hotels:** (We try to leverage this information to better accommodate our customers)

- Name of the hotel \_\_\_\_\_ Best hotel I’ve stayed at.. ☐ ☐ ☐ ☐ ☐ ☐
- Was this hotel recommended during your registration process?..... ☐ YES ☐ NO
- Problem? (brief description) \_\_\_\_\_

**SEE BACK**



Evaluation - Courseware  
Post Building Techniques V5.0, Course #MT11060

Please share your opinion for all of the following sections with a "check" in the appropriate box

**Material:**

	STRONGLY DISAGREE	DISAGREE	SOMEWHAT DISAGREE	SOMEWHAT AGREE	AGREE	STRONGLY AGREE
1. The training material supported the course and lesson objectives.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2. The training material contained all topics needed to complete the projects.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3. The training material provided clear and descriptive directions.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4. The training material was easy to read and understand.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5. The course flowed in a logical and meaningful manner.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6. How appropriate was the length of the course relative to the material?.....	<input type="checkbox"/> Too short <input type="checkbox"/> Too long <input type="checkbox"/> Just right					

Comments on Course and Material: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

Overall impression of course.....Poor ☐ ☐ ☐ ☐ ☐ ☐ Excellent

\_\_\_\_\_

\_\_\_\_\_

**Student:**

1. I met the prerequisites for the class (I had the skills I needed).....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2. My objectives were consistent with the course objectives.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3. I will be able to use the skills I have learned on my job.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4. My expectations for this course were met.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5. I am confident that with practice I will become proficient.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Name (optional): \_\_\_\_\_ Location/room \_\_\_\_\_

☐ Please "check" this box if you would like your comments featured in our training publications.  
(Your name is required at the bottom of this form)

☐ Please "check" this box if you would like to receive more information on our other courses and services.  
(Your name is required at the bottom of this form)

*Thank you for your business. We hope to continue to provide your training and personal development for the future.*