

UGS CONNECTION



AMERICAS 2008



Siemens PLM Software

SIEMENS

Advanced Post Builder Techniques

2008



My Contact Info

- ▶ Ken Akerboom
Moog, Inc
East Aurora, NY
kakerboom@moog.com
- ▶ Specialty Engineered Automation
akerboom@sea4ug.com

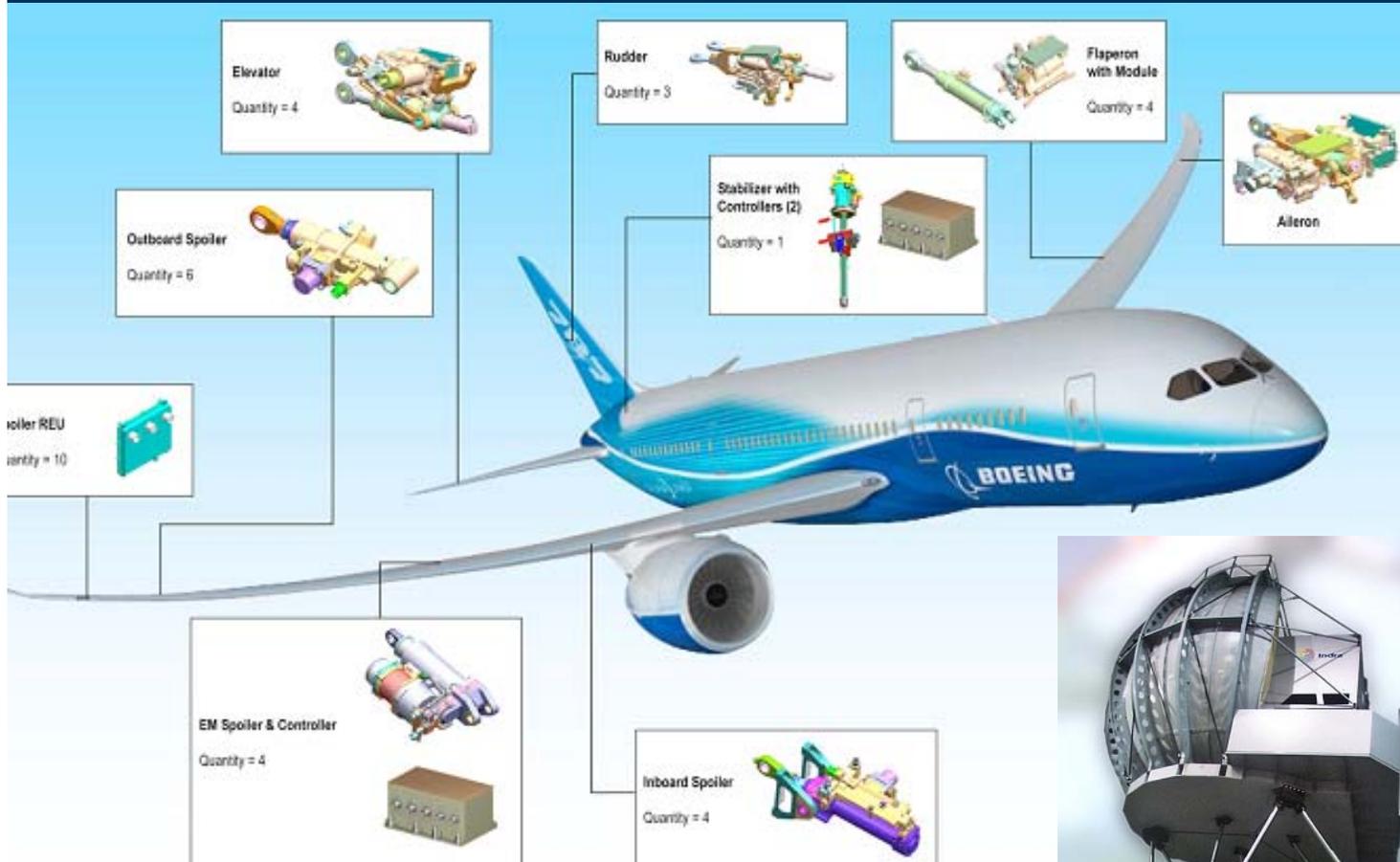
Who is Moog?

UGS
CONNECTION

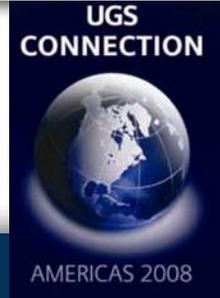


AMERICAS 2008

- ▶ We don't make musical instruments or auto parts...



Topics



- ▶ “source”ing in tcl files
- ▶ Replacing/modifying Post Builder “proc”s with your own
- ▶ Using a UGPost post to create “shop docs” without using Shop Docs
- ▶ The good, the bad, the ugly, and the %#&^!!\$*&^!
- ▶ NOTE: to simplify the display, most tcl code in this presentation does not have required “global” variable definition, nor does it have required (or at least highly recommended) error checking...

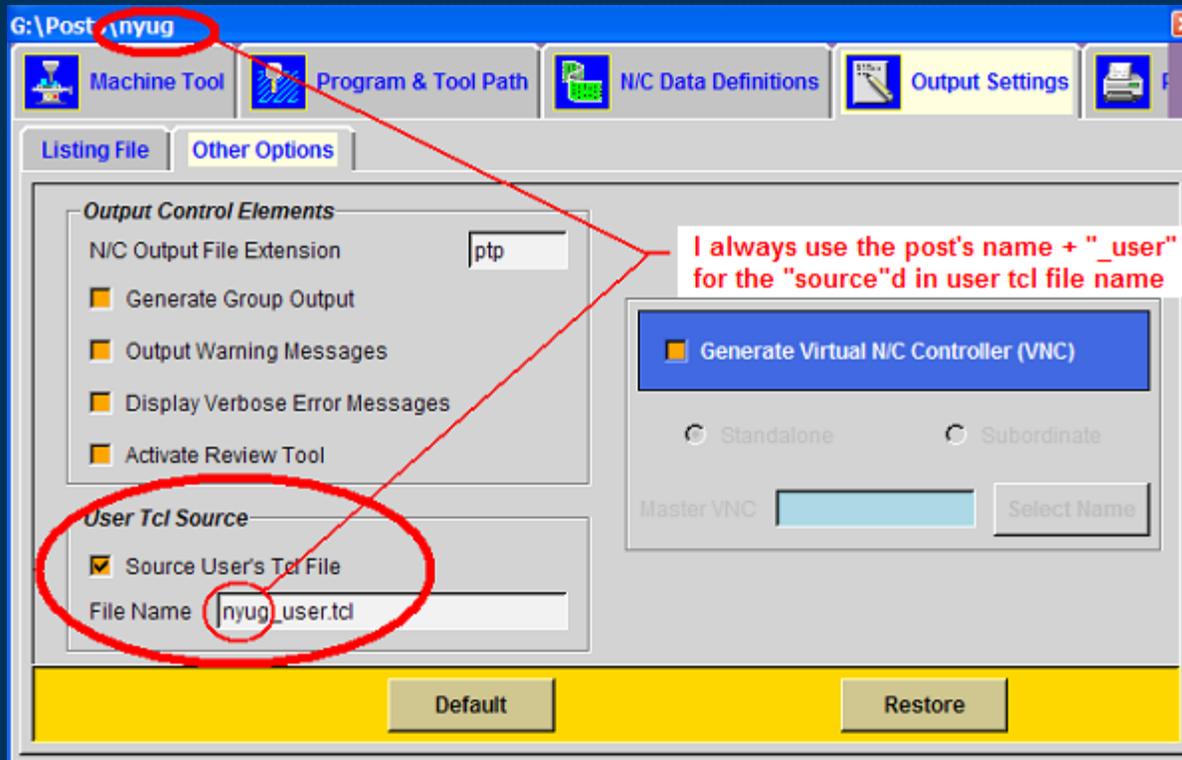


What is “source”ing?

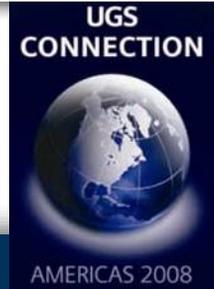
- ▶ You can use the tcl “source” command to include data and/or procs from another tcl file into the current file:

```
source "${cam_post_dir}Moog_common/moog_common.tcl
```
- ▶ One way to think of “source” is:
When the tcl interpreter encounters a “source” command, it copies the text from the “sourced” file and pastes it into the current file in place of the “source” command.
- ▶ Note you can’t include stuff selectively
 - ▶ You get all of the file.
- ▶ In a PB post, you don’t edit the Post Builder created tcl file, instead...

Sourcing in a "User" tcl file



By default, this MUST be in the same folder as the post...



What to put into the “user” tcl file?

- ▶ Extra variable definitions: `mom_sys_coolant_code(THRU-LP)`
- ▶ Event handlers for UDEs
- ▶ Replace or modify procs created by Post Builder
- ▶ Complex code from custom commands (PB_CMD_*)
- ▶ Code used in multiple custom commands
- ▶ Linked posts can share the same “user” tcl file to share common code between the posts
- ▶ You can also source in common code shared between multiple post processors

“source” in the *_User.tcl file

Examples from Moog:

- ▶ Procs common to all posts:

```
source "${cam_post_dir}Moog_common/moog_common.tcl"
```

- ▶ Procs common to a machine type:

```
source "${cam_post_dir}Moog_common/okuma_common.tcl"
```

- ▶ Procs common to a department:

```
source "${cam_post_dir}Moog_common/okuma_sbc_common.tcl"
```

- ▶ Procs related to a specific post feature

```
source "${cam_post_dir}Moog_common/thread_mill_macro_okuma.tcl"
```



“source” in the Post Builder’s *.tcl file

- ▶ In the .tcl file created by Post Builder, it sources in 2 files:

- ▶ Near the beginning, it sources in ugpost_base.tcl:

```
source ${cam_post_dir}ugpost_base.tcl
```

- ▶ Near the end, IF IT EXISTS, it source’s in the “user” tcl file:

```
set user_tcl_file ${cam_post_dir}nyug_user.tcl
if { [file exists $user_tcl_file] } {
    source $user_tcl_file
}
```



Who Wins?

- ▶ If a variable or proc is defined multiple times, which definition is used?
- ▶ In tcl, the LAST* definition will be used, if you read through the tcl file(s) from start to end, “source”ing in other files at the location of the “source”.
- ▶ * Note that some procs are defined by the post running a PB_CMD_* that actually defines the proc, so the proc does not get defined as the tcl interpreter is reading the tcl file (search for “uplevel” in the PB .tcl file). Rather it is defined AFTER all the tcl files are read and the post is actually run. E.g.:
`MOM_rotate{}` is not defined until `PB_CMD_kin_init_rotary{}` is run.

Who Wins (cont.)?

This:

post.tcl

```

proc A {} {}
proc B {} {}
proc C {} {}
proc D {} {}
source p_user.tcl
    
```

post_user.tcl

```

source com.tcl
proc A {} {}
proc D {} {}
proc E {} {}
proc G {} {}
    
```

com.tcl

```

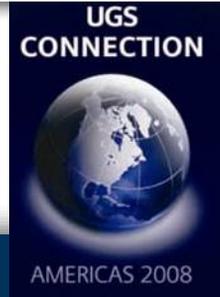
proc A {} {}
proc C {} {}
proc E {} {}
proc F {} {}
    
```

Is the same as:

```

proc A {} {}
proc B {} {}
proc C {} {}
proc D {} {}
proc A {} {}
proc C {} {}
proc E {} {}
proc F {} {}
proc A {} {}
proc D {} {}
proc E {} {}
proc G {} {}
    
```

“proc”s in **RED** (LAST one found)
will be used



Where to source in other files

- ▶ My preference is to source other files in at the BEGINNING of the *_user.tcl file
- ▶ As the *_user.tcl file is sourced at the end of the Post Builder .tcl file, this means...
- ▶ Anything YOU write will over-ride the UGS post-builder created code
 - ▶ If they have a bug, or don't do what you want, you don't have to edit the post's tcl file every time you save it.
- ▶ By “source”ing your other (shared) files at the beginning of the *_user.tcl
 - ▶ If a post need to over-ride one or two procs in the common (sourced-in) tcl files, you can add that code AFTER they are sourced in, but still use the rest of the common code.



Replacing PB's procs – “normal” procs

- ▶ If you want to completely replace a PB generated proc...
- ▶ If the proc is “normally” defined...
 - ▶ Just put a “proc MOM_linear_move {} {...}” in your user .tcl file

The PB proc is created by another PB proc

- ▶ There are a couple ways, but you have to write your own PB_CMD_* to run AFTER the PB internal proc. You can either:
 - ▶ Use “uplevel” like PB does (emulate how PB defines the proc)
 - ▶ Use the tcl “rename” command (twice)
 - ▶ Create your proc with a slightly different name in the *_user.tcl file (e.g. “proc Moog_mom_rotate”)
 - ▶ In the PB_CMD_Moog_*
 - ▶ “rename” the UGS procedure:
`rename MOM_rotate UGS_MOM_rotate`
 - ▶ And “rename” yours to the desired name:
`rename Moog_mom_rotate MOM_rotate`

If you want to alter the PB proc's behavior

- ▶ E.g. for MOM_text, you want to force everything to UPPERCASE

- ▶ In the *_user.tcl file:

```
# make sure we haven't already renamed it
if {![string length [info procs UGS_MOM_text ]]} {
    rename MOM_text UGS_MOM_text
}
```

```
# Now define our own proc, call the PB one at the end:
```

```
proc MOM_text {} {
    #... Do uppercase here ...

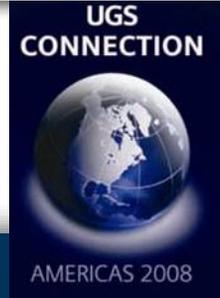
    # Now call the UGS proc
    UGS_MOM_text
}
```

Documentation without Shop Docs



- ▶ While you can use the Siemens “Shop Docs” module to create your shop floor documentation...
 - ▶ It’s a different architecture and syntax to learn from Post Builder
 - ▶ If you’re not careful, it includes ops/tools/etc. that aren’t output in the posted code
 - ▶ It’s run as a separate action in NX, so it’s easy for users to forget to re-run after changing a program
 - ▶ While it can create graphical images, they are of dubious value in many parts (only one layout/orientation/zoom/etc.)
 - ▶ If you need different variations for different machines (e.g. mill vs. lathe) you may need different Shop Docs.

Using UGPost to create your shop floor documentation



- ▶ My personal preference is to create the documentation files as a part of the posting process.
 - ▶ Always captures ONLY what is in the posted code
 - ▶ Always consistent with the posted code
 - ▶ Graphics can be created using an API extension to Ugpost
 - ▶ The way I wrote my code, the same shop docs code is shared by ALL my posts
 - ▶ The output changes based on the information saved, which can be tailored by what is saved while posting.
 - ▶ You can use this method to add tool lists, etc. to the beginning of the posted code, as well as outputting separate files.

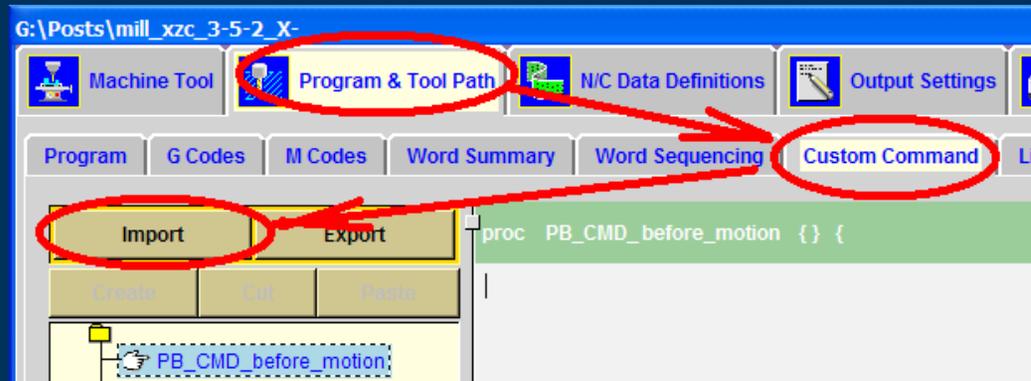


UGPost docs – general architecture

- ▶ What I do is:
- ▶ Store all the information, as I encounter it, in tcl lists and arrays:
 - ▶ Operation info in “start of path” event
 - ▶ Tool info in “Tool change” and “first move” events
 - ▶ I use part, operation, and tool attributes to store “other” info (descriptions, fixtures, etc.) in the part so the post can access it.
 - ▶ Save Cutter Dia. Comp. registers used in “cutcom on” event
- ▶ Array indexes: Use strings that NX forces to be unique:
 - ▶ Operation names
 - ▶ Tool Names
 - ▶ Index variables you create yourself
- ▶ Once the post is done, in the “end of program” event, I run a proc that actually creates the documentation file.

Architecture (cont)

- ▶ Again, all the actual tcl code to save/output the docs is in a single “source”d in tcl file.
- ▶ I also have a set of custom commands that I “Import”, then add to the appropriate events:



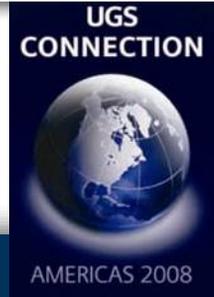
- ▶ Typical custom command looks like:


```
#
# Saves tool info
# Place in "Initial Move" and "First Move"
#
    Moog_SD_Save_Tool_Info
```
- ▶ Adding docs output to a post takes me 5 minutes



EXAMPLE: Saving tool data

```
proc PLM_SD_Save_Tool_Info {} {;# globals omitted for brevity
  if {![hiset mom_tool_name]} { return }
  set plm_sd_tool_no($mom_tool_name) $mom_tool_number
  if {[hiset mom_tool_adjust_register]} {
    # this stores ONE value - You will probably want to store
    # all values, then sort/uniqueify in output routine
    set plm_sd_tl_adj($mom_tool_name) "$mom_tool_adjust_register"
  }
  if {[hiset mom_tool_catalog_number]} {
    set plm_sd_tl_catno($mom_tool_name) "$mom_tool_catalog_number"
  }
  if {[hiset mom_attr_TOOL_DESCR]} {
    set plm_sd_tool_descr($mom_tool_name) "$mom_attr_TOOL_DESCR"
    unset mom_attr_TOOL_DESCR      ;# # clear this variable here
  }
}
```

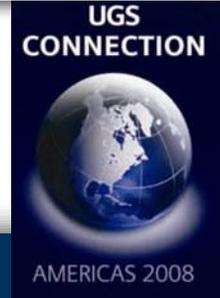


Warning about tool info

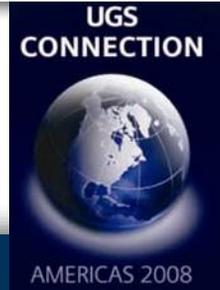
- ▶ UG post does NOT clear tool info between operations.
 - ▶ So if the first tool defines a catalog number, and the second does not, `mom_tool_catalog_number` will still exist when you save tool info for the second tool
- ▶ I wrote a proc (put in “end of path”) to clear MOST tool info (some you do want to keep) [note “global” lines omitted for brevity]:

```
if {![hiset mom_next_oper_has_tool_change]} { return }
if {$mom_next_oper_has_tool_change != "YES"} { return }
set global_vars [lsort [info globals "mom_tool*"]]
foreach global_var $global_vars {
    # there are a few variables we DO NOT want to clear...
    if {![string compare $global_var "mom_tool_use"]} { continue }
    if {![string compare $global_var "mom_tool_count"]} { continue }
    upvar #0 $global_var global_val
    unset global_val
}
catch { unset mom_tracking_point_name }
catch { unset mom_tracking_point_type }
```

EXAMPLE: Adding a tool list to the beginning of an NC file

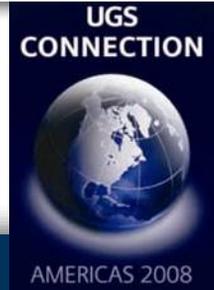


- ▶ Use Ugpost to write a “flag line” where you want the tool list at the beginning of the file
 - ▶ Make it some unique text like
“TOOL_LIST_GOES_HERE”
- ▶ Like “UGPost Docs”, save the tool information (in arrays based on the tool name) when you encounter it (e.g. in tool change events)
- ▶ In the “End of program” event have a proc do the following (next slide)



Adding a tool list to a posted file (cont.)

- ▶ Close the current output file using `MOM_close_output_file`
- ▶ Rename the output file to a temp name
 - ▶ This and the rest of the steps use “normal” tcl commands
- ▶ Open the “temp” file, and also open a new file using the desired output filename
- ▶ Read through the temp file, line by line
 - ▶ If not the “flag” line, just write the line to the output file
 - ▶ If it IS the “flag” line, write the tool list instead
- ▶ Close both files
- ▶ Delete the temp file



EXAMPLE: End of file code

```
proc NYUG_Insert_Tool_List {} {;# globals omitted for brevity
  if {![array size plm_sd_tool_no]} { return}
  # extract tool numbers once
  foreach i [array names plm_sd_tool_no] { lappend ToolNos $plm_sd_tool_no($i) }
  set ToolNosSorted [lsort -integer $ToolNos]
  set last_tool ""
  set tool_count [llength $ToolNosSorted ]
  MOM_close_output_file $mom_output_file_full_name ; # Close output file

  # rename current output file, so file with tool list is the desired name.
  file rename -force $mom_output_file_full_name ${mom_output_file_full_name}_old

  # open both files
  set OLD_File [open ${mom_output_file_full_name}_old r]
  set NEW_File [open $mom_output_file_full_name w]
```

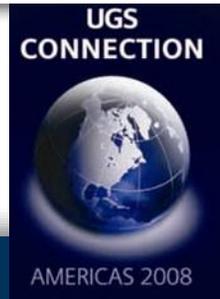
...cont...



EXAMPLE: End of file code (cont.)

```
# read thru old file
while {[gets $OLD_File line] >= 0} {
    # if line is flag where to put tool list
    if {$line == "PUT_TOOL_LIST_HERE"}
        # output tool list in new file
        for {set i 0} {$i < $tool_count} {incr i} {
            if {$last_tool == [lindex $ToolNosSorted $i]} { continue }
            set last_tool [lindex $ToolNosSorted $i]
            # find tool name(s) for that index
            foreach tname [array names plm_sd_tool_no] {
                if {$plm_sd_tool_no($tname) == $last_tool} { break }
            }
            # actual line of data
            puts $NEW_File "[format "%3d" $last_tool] $ tname ..."
        }
    } else {
        ;# otherwise, just copy the line
        puts $NEW_File $line
    }
}...cont...
```

Finishing...



```
} ;# end of "while" loop
close $OLD_File
close $NEW_File
file delete -force ${mom_output_file_full_name}_old
}
```

Output to a separate file

- ▶ Similar to what I just showed you, but
 - ▶ You don't need to play with the posted code file
 - ▶ Just open the desired output file, write to it, and close it

Process Instructions

This File:	MYG0318.HTML
Created By:	kakerboo
Date:	07/25/2006 15:26
Part Name:	MYG0318
Part Rev:	B

Path	Tool No	Description	Hole #	"C" Axis Deg.	SFM	IPR	RPM	IPM	Path Time
HEADER									0:00
DLZ_10	113	TEST DESCRIPTION	1-2	0.0			800	1.0	0:53
RT_20	103				375	0.0120			2:14
RF_30	103				225	0.0080			1:25
RF40-EDGE	103					0.0080	225		1:18
Total Time									2:02:10

Tool Information				
Tool No	Length Reg.	Dia. Reg.	Tool Name	Stock Number
3	39		P181070-0004	P181070-0004
9	99		P181073-0010	
	99		P181073-0010A	
55	55	55	P181115-0078	P181115-0078

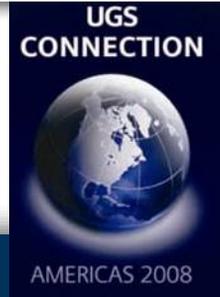


The good

- ▶ By using “source” and the *_user.tcl file, you can easily extend the powers of PB and share code between posts
- ▶ If you don’t like how a PB “proc” works, you can change it (e.g. 5 axis mill rotary axis clamping around JUST the line with the rotary axis move, not the entire rapid event).
- ▶ This is some very powerful stuff

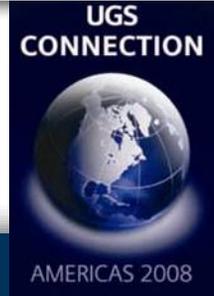
The bad

- ▶ PB takes longer and loooonger and loooooonger to save every time you save
 - ▶ If you do most of your coding in a sourced tcl file, you eliminate a lot of saves in PB (and text editors save FAST)
- ▶ Don't play with PB created procs unless you REALLY know what you are doing
 - ▶ I've "shot myself in the foot" many times, but that's hw you learn
- ▶ You have to be careful when altering procs shared between posts – you may fix one post but break another.



The ugly

- ▶ I'm getting tired of not having mom variables for parameters I need in my posts...
- ▶ Or having to use undocumented ones...
- ▶ Or having variables that USED to work get completely hosed (lathe tracking points in NX4...)



The %#&^!!\$*&^!

- ▶ I _used_ to recommend writing all sorts of TK dialogs to prompt users for stuff.
- ▶ Unfortunately, if you have a multi-processor system (I know 64 bit multi-core CPU Windows machines have this problem, both WinXP32 and WinXP64)...
 - ▶ The version of tcl Siemens uses (v8.1) is so old that it has a bug that will randomly lock your NX session (I.e. you have to use task manager to kill it). I haven't found a reliable workaround.
 - ▶ I tried using `MOM_run_user_function` and an API program instead of using TK dialogs, but this also locked NX up (although in a different way)
- ▶ If you use an API program to deal with the dialogs, then have it post the file, its ****should**** work. (but now you need toolbars or Menuscript or ... to run the API program)



A plea (or 2 or 3) to Siemens:

- ▶ PLEASE get your MOM tcl extensions up to a current version of tcl...
- ▶ When you re-base MOM on a newer version of tcl....
 - ▶ PLEASE include TK (or other GUI toolkit of your choice) so we don't have to jump through hoops adding dialogs to our posts...
- ▶ PLEASE convert XZC mill posts to use the regular kinematics module so "head" objects and ... will work correctly.

UGS CONNECTION



AMERICAS 2008



Siemens PLM Software

SIEMENS

Hope this was interesting...
Any questions?

2008