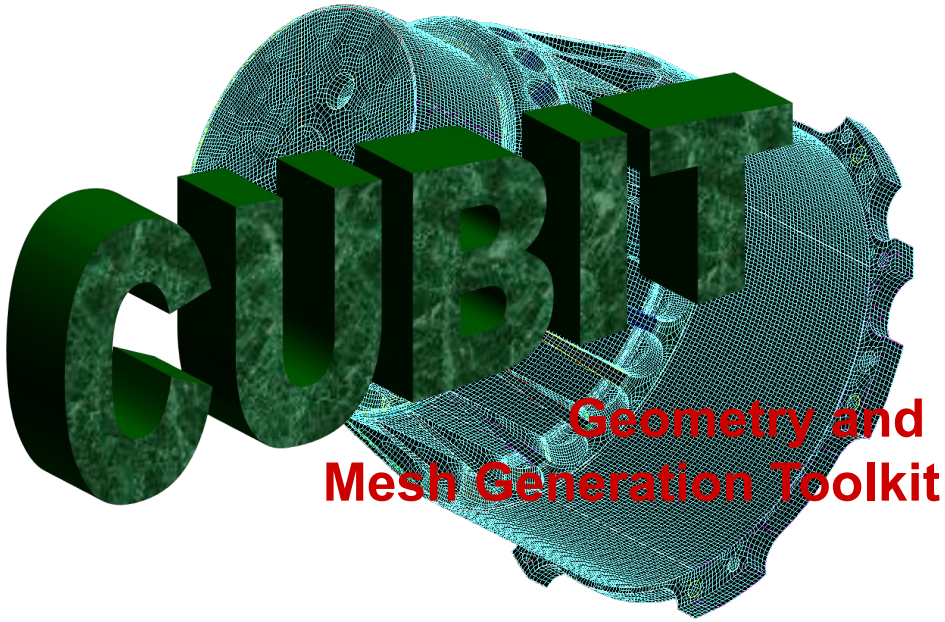


CUBIT Fast-Start Tutorial

15. Customization and Scripting

Cubit Command Language

- **Cubit is controlled by a command language.**
  - This command language is generated by the GUI and passed into the Cubit processor.
  - The commands can be journaled to a file to provide a history.
  - The commands are echoed to the command window in the GUI.
  - User help via email is frequently given in terms of Cubit commands.
  - Cubit can be programmed via this language.
- **It is VERY helpful to understand the syntax and power of the Cubit command language.**

Cubit Command Syntax

- Case is not significant.
- Commands may be abbreviated to the smallest unique set of characters.
- A hash mark (#) denotes a comment.
- Lines may be continued to the next line with a backslash (\) at the end of the continuing line.

Cubit Command Syntax

- **Commands are typically verb noun format.**
  - mesh volume 1
  - Mesh is the verb acting on volume 1.
- **However . . . the verb may be optional**
  - cylinder radius 2 height 10
  - This is really
    - **create cylinder radius 2 height 10**
    - **The create keyword is optional.**
- **Or . . . the syntax is just noun first**
  - surface 1 scheme pave
  - This is frequently used when setting an attribute on an entity.

Cubit Help

- **Help on any command can be obtained by typing a question mark (?) or an ampersand (&) at the command line prompt.**
  - The question provides completion options of a given command.
  - The ampersand gives matches of command words ignoring the order of the words.
- **You can also type help and give any command word for help on that command.**

Specifying Ranges

- **Cubit commands frequently take a range of ids and there are specific keywords that can be used in ranges.**
  - draw surface all
  - list volume 1 to 10 by 2
  - volume all except 3 scheme auto

Ranges by Topology Traversal

- **Ranges may also be specified by topological relations.**
  - Draw vertex in volume 1
  - List volume in vertex 1 and 10
  - Curve in surf 2 size .3
- **If there is a range of entities on both sides of the "in" keyword the result is the intersection of the ranges.**
  - curve 1 to 3 in body 4 to 8 by 2
- **Topology may include mesh entities**
  - draw node in surface 3
  - draw surface in node 3

Specifying by Criteria

- **Cubit allows specification by a given criteria using the "with" keyword.**
  - {Entity_Type} With {Criteria}
  - draw curve with length <1
  - locate surface with is_meshed = false
  - highlight volume with not is_virtual
- **The boolean operators "and" "or" control how statements are combined.**
  - node with x_coord > 10 And y_coord > 0
- **The "of" keyword returns the value of a single entity for comparison**
  - list curve with length < length of curve 5 ids

Operators on Criteria

- **Boolean operators**
  - = or ==
  - <=, >=, <, >
  - <> not equal
    - **draw surface with x_max <= 3**
    - **draw volume with z_max <>12.**
- **Arithmetic operators**
  - +, -, *, /
    - **draw surface with length * 3 + 1.2 > 10**

Criterion Functions

ID	ID of an entity
Length	The length of a curve or edge
Is_Meshed	Whether an entity is meshed
Is_Spline	Whether an entity is a NURBS
Dimension	Topological dimension of entity
X_Coord, Y_Coord, Z_Coord	The x, y, or z coordinate of the centroid of an entities bounding box.
X_Min, Y_Min, Z_Min	The x, y, or z coordinate of the minimum extent of the entity's bounding box
Is_Merged	If the entity is merged

Criterion Functions

Is_Virtual	Whether an entity is virtual geometry.
Has_Virtual	An entity "has_virtual" if it is virtual itself, or has at least one child virtual entity
Is_Real	An entity "is_real" if it has at least one real (non-virtual) entity.
Num_Parents	specify geometry entities with a specified number of parent entities. May be used to find "free curves" where num_parents=0 or non-manifold curves where num_parents>2.

Example 1

bri x 10

cyl radius 2 z 12

subtract vol 2 from vol 1

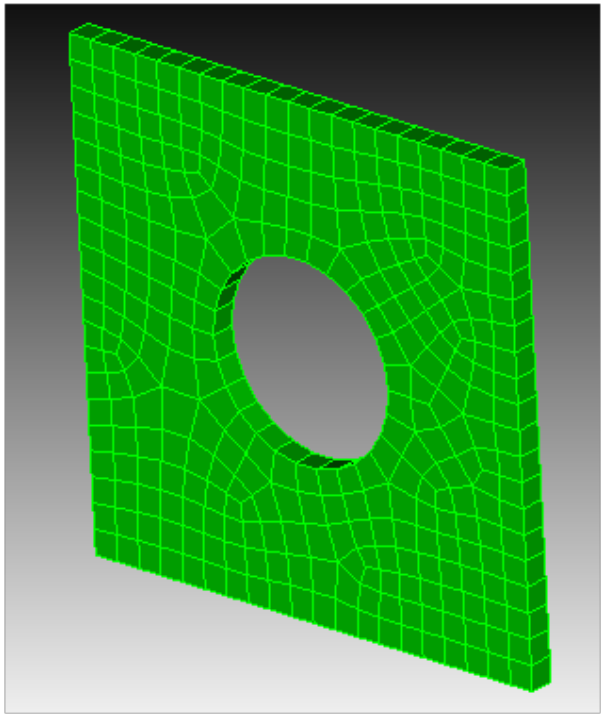
vol 1 size .5

vol 1 scheme auto

mesh vol 1

draw surf in vert 1

draw hex in node in surf 11



Introduction to Aprepro

- **Aprepro = Algebraic Pre-Processor**
- **A built-in miniature programming language**

- **Purposes**
  - Parameterize a journal file
  - Error checking
  - Other control logic

Basic Aprepro Syntax

- **Aprepro expressions are wrapped in curly braces**
- **Aprepro evaluated first, results inserted into command:**

```
Brick X {10}
```

And

```
Brick X {5*2}
```

Are Equivalent To

```
Brick X 10
```

Aprepro Variables

- **Variables are named values**
- **Names are case sensitive**
- **Variable type is Number or String**
- **Defined in a comment**

```
# {width=2}
# {r="radius"}
```

- **Use anywhere in a command, as if you typed the variable's value:**

```
brick x {width}
cylinder height 5 {r} 10
```

Aprepro Equations

- **Variable values can be changed**

```
# {x = 1}
# {x = 5}  ##This will give you a warning
# {x++}  ## Increase value of x by one, no warning
```

  - To avoid warnings, make variable name start with an underscore (_), or use ++ and --

- **Convenient way to see variable value: *comment* command**

```
Comment x
User Comment: 6

Comment "x is" x "and y is" y
User Comment: x is 6 and y is <undefined>
```

Operators

- **Addition: +**
- **Subtraction: -**
- **Multiplication: ***
- **Division: /**
- **Power of: ^ ({3^2} = 9)**

- **Math expressions can be used just about anywhere:**

```
# {width=3}
# {width_squared = width^2}
brick x {width} y {width*2} z {width_squared}
```

Aprepro Functions

- **Functions calculate or look up values**
- **Function name followed by parameters in parentheses**
- **The number and type of parameters depends on the function**
- **Commas between parameters**
- **Parameters can be constants, variables, or equations**

```
# {errs = get_error_count()}
# {x=cos(PI/2)}
# {vertex_x = Vx(30)}
# {random_number = rand(10, 20)}
# {Print ("Hello World")}
```

Types of Functions

- **Math Functions**
  - sin(num), cos(num), asin(num), etc…
  - sqrt(num), exp(num), log(num), ln(num), etc…
- **String Manipulation Functions**
  - Quote(string), toupper(string), tolower(string)
- **Utility Functions**
  - Print(string), PrintError(string)
  - FileExists(string), HasFeature(string)

Types of Functions

- **Session Information Functions**
  - NumVolumes(), NumSurfaces(), etc…
  - get_error_count(), set_error_count(num)

- **Entity Information Functions**
  - Vx(num), Nz(num)
  - Radius(num), SurfaceArea(num), Length(num)
  - CurveAt(num, num, num), HexAt(num, num, num)

Flow Control

- **Three types of flow control**
  - If statements
  - Loops
  - Include files

If Statements

- **If true then do**

```
#{if (my_variable < 3)}
  brick x 3
#{else}
  brick x {my_variable}
#{endif}
```

- **If defined and not zero then do**

```
#{ifdef(make_a_brick)}
  brick x 3
#{endif}
```

- **If zero or not defined then do**

```
#{ifndef(skip_the_brick)}
  brick x 3
#{endif}
```

Loops

- **Repeat a set of commands some number of times**

```
# Create 3 bricks
#{Loop(3)}
  brick x 10
#{EndLoop}
```

- **Loop statement accepts numbers or variables, but not expressions**

```
# {Loop(3)}  #OK
# {Loop(x)}  #OK
# {Loop(x-1)}  #error
```

Example 1 – Accurate Coordinates

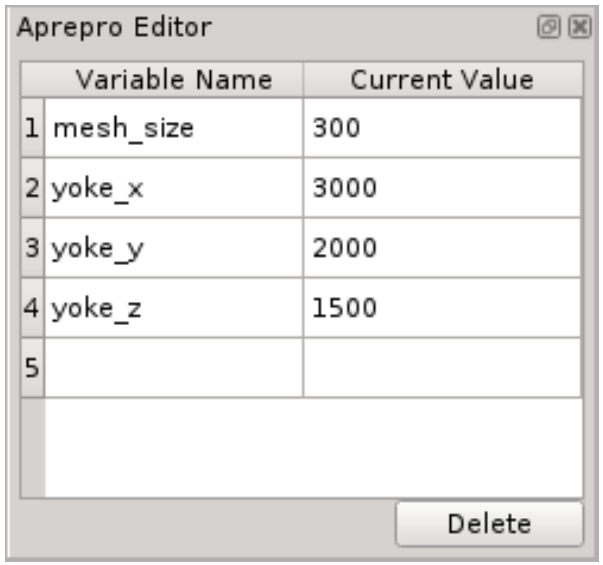
Create Vertex {Vx(1) + 10} {Vy(1)} {Vz(1)}

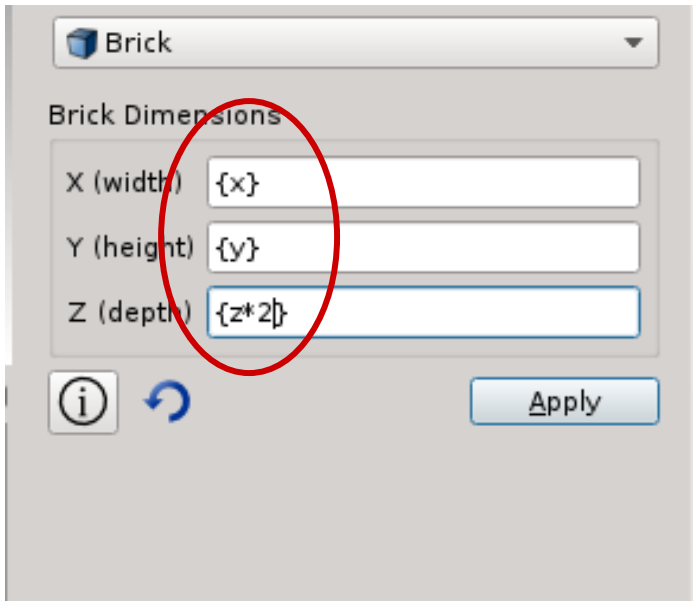
Example 2 – Name that Volume


```
Create Brick Width 1
Create Brick Width 1
Volume 2 move x 1
Volume {Id("volume")-1} Name "Martha"
Volume {Id("volume")} Name "George"
```

Aprepro in the GUI

- **Use aprepro anywhere in the GUI, using curly braces**
- **See and modify aprepro variables in the power tools**





Example 3 - Write Head

- Play ‘write_head.jou’
- Open the file in the journal editor
- Change the values of yoke_x, yoke_y, yoke_z
- Play the journal file again.
- Set hex_mesh = 1
- Play again
- Set up a new variable for the return pole z
- Why doesn’t everything move correctly?