CFD EXPERTS

Simulate the Future

WWW.CFDEXPERTS.NET

Ansys
2021 R2
Engineering What's Ahead.

Ansys ICEM CFD Programmer's Guide

ANSYS, Inc.
Southpointe
2600 Ansys Drive
Canonsburg, PA 15317
ansysinfo@ansys.com
http://www.ansys.com
(T) 724-746-3304
(F) 724-514-9494

Release 2021 R2
July 2021

Table of Contents

Conventions for Using This Manual 1
ICEM CFD Scripting Basics 3
    Scripting Commands 9
    Stand-alone scripting 10
Form Creation Functions 13
External Commands 23
Translation Functions 27
Display Functions 33
Geometry Functions 35
Blocking Commands 113
Meshing Functions 139
    Meshing Directives 148
    Structured Mesh Editing and Modification Functions 153
    Unstructured Mesh Editing and Modification Functions 160
    Cartesian Mesh Editing and Modification Functions 188
    Miscellaneous Mesh Editor Functions 191
Output Functions 193
    Boundary Condition Editing and Modification Functions 195
Utility Functions 203
    IC3M Functions 206
    XML Report Editing and Modification Functions 207
    Regression Testing Functions 207
Workbench Integration 209

Conventions for Using This Manual

To maximize the benefit of this manual, you should download the scripts and case files available on the Ansys customer site.

1. Create a folder named **`Samples`** in the Ansys installation directory under `../v212/icemcfd/`
2. To access documentation files, go to the documentation area of the customer site.
3. Follow the link labeled **Customer Portal** near the top of the screen. You may have to log in.
4. Use the search box to find **`ICEM CFD Programmer's Guide files`**.
5. Follow the link labeled **Documentation:: Script files and input files for the ICEM CFD Programmer's Guide**.
6. Click the link to download the `ICEM-CFD-Programmers-Guide-files.zip` archive.
7. Unzip the archive into the **`Samples`** folder created earlier.

   The final path to the scripts and case files will be `../v212/icemcfd/Samples/ProgrammersGuide/`

Note:

To get results when running example scripts in batch mode, you will need to create a second folder named `output`, parallel to the folder named `input` in your `../Samples/ProgrammersGuide` folder.

Tip:

In this manual, default arguments for each **proc** argument, if any, are given in [] brackets.

Library Functions

In addition to the scripting functions described in subsequent chapters, ICEM CFD includes several output library functions. Descriptions of the output library functions are in your installation directory in `..\win64_amd\help\library` and are summarized below:

- `blocking_doc.html`: Describes the format and structure for the blocking file (*.blk).
- `bocolib.html`: The boundary condition library contains routines to access the information in the boundary condition files.

- `domainlib.html`: The domain library contains routines for creating, modifying, and reading information in domain files - nodes (locations), elements (volumes, shells, or trusses), and subdomains.
- `tetinlib.html`: The tetin library file contains descriptions of commands and arguments which serve to define the CAD geometry and the parameters for the unstructured mesh generation.
- `topolib.html`: The topology library contains routines to access the topological information characterizing structured meshes such as number of blocks, block size and block-to-block connectivity.

ICEM CFD Scripting Basics

The **Message Window** in Ansys ICEM CFD is a **Tcl/Tk shell** with an extended library consisting of Ansys ICEM CFD commands. Most of these commands begin with "**ic_**" to indicate an Ansys ICEM CFD command. Documentation for **Tcl**/**Tk** procedures can be obtained at http://www.tcl.tk.com. Many books have also been written on the subject.

Tcl, which stands for Tool Command Language, is a string-based programming language. This means that, except for inside the **expr** command, you need not discern between variable types, such as integers, doubles, or strings. Everything is treated as a string. There is also no need to set aside memory (dimension) for variables, and they can be reset to values of any length without re-dimensioning.

Tk stands for Toolkit and contains the add-on **Tcl** commands that allow graphical interfaces or windows of an application to be made.

Command Syntax

The basic syntax for a **Tcl** command is:

command arg1 arg2 arg3 ...

Commands are terminated by either a newline character (not usually displayed) or a semicolon, "**;**".

Spaces are key characters which separate arguments.

Commands can only be continued to the next line with a "****" at the end of the line, or with the beginning of an argument grouping using "**{**" or **"**.

Argument Grouping

If a space is to occur inside an argument, then the argument must be grouped as one so that it is not interpreted as separate arguments. For example, the syntax for the **puts** command is:

puts ?**-nonewline**? ?*channelId*? *string*

The text between question marks, "**?...?**" includes optional arguments or switches which may or may not be present. Consider the following commands and their return values:

```
    %puts {Hello world}
Hello world
```

```
%puts Hello world

cannot find channel named "Hello"
```

In the first example, note that the grouping brackets, "**{...}**", were stripped out by the interpreter, but it was treated as a single argument.

In the second example, the arguments, "Hello" and "world" are treated as separate arguments where "Hello" is the channelId. At this point, no channel named "Hello" exists.

There are 3 types of grouping:

- **{...}** - Groups without allowing substitution inside
- **"..."** - Groups while allowing substitution inside.
- **[...]** - Interrupts the current command to do the command contained inside the brackets. The next word following "**[**" must be a command. This creates a nested **Tcl** command structure. Some **Tcl** documentations do not consider this a grouping structure.

Note:

The **puts** command will not output text to the Ansys ICEM CFD message window. To do this, use **mess** or **ic_mess**. This command automatically does the **-nonewline** switch. You must use "**\n**" inside the message string to insert a new line.

Substitution

There are 3 types of substitution:

- **Command** - This is done whenever there are square brackets, "**[...]**". The command is executed and the return value is substituted before continuing with the original command.
- **Variable** - Variable substitutions are done by using a dollar sign, "**$**", in front of the variable.
- **Backslash** - When a backslash, "\" is encountered, the next character is treated special. For example, "**\n**" means to substitute a new line character.

The following commands can be used to set a value to a variable and then substitute that value for the variable later:

```
%set x 5

5
%mess "x is: $x\n"
x is: 5
%mess {x is: $x\n}
x is: $x\n
```

Here, the "**\n**" is used to ensure that the next text printed to the message window will appear on the next line. Without it, the text will continue on the same line. Also note that the **set** command will also return the value as well as set the variable to the value.

In the third example, note that no substitution, variable or backslash, was done inside the brackets, "**{...}**".

Note:

Argument grouping is ALWAYS done before substitution, so the number of arguments input into a command is always determined before any variable or command substitutions that may result in multiple return values.

Creating a Simple Script

The following examples demonstrate how to load multiple geometry files of varying names and save them all as one new geometry file, without using the Ansys ICEM CFD window (GUI) displayed (GUI stands for Graphical User Interface):

Do the following in the Ansys ICEM CFD window:

1. Go to **File > Replay Scripts > Replay Control**, then load the tetin file.

2. Save the tetin file to a different name.

You should see the following commands printed:

```
1. ic_load_tetin support/test/tetin1.tin
2. ic_geo_get_body_matlpnt LIVE.0
3. ic_boco_solver
4. ic_boco_clear_icons
5. ic_boco_natran_csystem reset
6. ic_uns_set_modified 0
7. ic_geo_set_modified 0
8. ic_undo_start
9. ic_save_tetin support/test/new_tetin.tin 0 0
```

You may discard all but the first and last commands since you are not interested in any GUI updates, boco settings, or the undo log. This is what remains:

```
 1. ic_load_tetin support/test/tetin1.tin
9. ic_save_tetin support/test/new_tetin.tin 0 0
```

The file name paths start from either the working directory as in this case, or from the top level drive. Since the tetin names can be changed, as well as the working directory, you must use variables. It is safe to specify all names from the top level, so set a variable that is the path of the working directory:

```
set wdir D:/user/support/test
set tetin tetin1.tin
ic_load_tetin $wdir/$tetin
ic_save_tetin $wdir/new_$tetin.tin 0 0
```

Note:

You can use "/" on Windows operating systems as well as Unix. The "\" is an escape character which interprets the next character differently. There are situations, however, where this could potentially cause problems, so it is a good practice to use the **file join** command. With the **file join** command inserted inside bracket expressions, the script looks like this:

```
set wdir D:/user/support/testset tetin tetin1.tinic_load_tetin [file join $wdir $tetin]ic_save_tetin [file join $wdir new_$te
```

Now, you can use a **foreach** loop to cycle through multiple tetin files. The simplest syntax for the **foreach** command is:

foreach *varname list body*

The **foreach** loop will step through a list of values assigning each next item in the string to the *varname*, and evaluating *body* each time. The *list* can be a proper **Tcl** list made using the **list** command, or by simply grouping text within **"..."** or **{...}**. List elements must be separated by spaces. With a **foreach** loop, the script becomes this:

```
set wdir D:/user/support/test
foreach tetin [list tetin1.tin tetin2.tin tetin3.tin] {
ic_load_tetin [file join $wdir $tetin]
}
ic_save_tetin [file join $wdir new_$tetin.tin] 0 0
```

The example illustrates another important property of grouping arguments with **{...}** or **"..."** (but not **[...]**). The grouping quotes or curly brackets (the "*body*" argument here) allow an argument to span multiple lines so that the current command is not terminated by the first newline character (which are usually not displayed in the text editor).

Now, you can save this script as `script.tcl` in the directory `D:/user`.

Running the Script

There are 4 ways to run the script:

- From the **Replay control** window - By using **Do one** or **Do all**
- Using the **File** > **Replay Scripts** > **Run from script file** option
- `source` - Using the **Tcl** command to execute **Tcl** commands in a file.

- `icemcfd -script` - Specifying a script to run when you start up Ansys ICEM CFD

All 4 methods can be used with the GUI displayed, but only the last two can be done in *batch* mode.

`source D:/user/script.tcl` can be typed in the message window, or Ansys ICEM CFD can be started in batch mode by adding the `-batch` argument to the Ansys ICEM CFD startup command (`icemcfd -batch`). Then you can **source** the script from the shell.

Using both the `-batch` and `-script` arguments (`icemcfd -batch -script D:/user/script.tcl`) will start Ansys ICEM CFD in batch mode and immediately run the script.

Note:

The `source` command can be used inside a script to execute **Tcl** commands from a separate script.

You can have Ansys ICEM CFD close immediately after your script is finished if you add the **exit** command to the end of your script.

Arrays and Environment Variables

Environment variables are accessed through a reserved global **array** variable named **env**.

An **array** is a variable name followed by another name within parentheses. This is a variable which contains sub-variables, each of which can have a value assigned. The following are array variables of the same array:

geom(point)
geom(curve)
geom(surface)

You can use the **env** variable to make the script portable between different installations of Ansys ICEM CFD by using it in the path for any scripting commands that run another executable inside Ansys ICEM CFD.

For example, the following line was printed in the **Replay control** window when a mesh was written to the CFX5 format:

ic_exec {C:/Program Files/Ansys Inc/v90/icemcfd/5.1-win/icemcfd/output-interfaces/cfx5} -dom D:/user/support/test/tetra_mesh.uns -b D:/user/support/test/family_boco.fbc -ascii -internal_faces D:/user/support/test/cfx5_input

where ***cfx5*** is the executable being run.

The **ICEM_ACN** environment variable points to the top Ansys ICEM CFD folder, 5.1-win. You can substitute the environment variable for the path leading up to 5.1-win. You can also set a working directory variable as before and substitute that where appropriate using the **file join** command:

set wdir D:/user/support/test

```
ic_exec [file join $env(ICEM_ACN) icemcfd output-interfaces cfx5] -dom [file join $wdir tetra_mesh.uns] -b [file join $wdir fami
```

Since argument grouping is always done before substitution, you need not put quotes, **"..."**, around **$env(ICEM_ACN)**. It is seen as one argument in the **file join** command even though its value contains spaces:

$env(ICEM_ACN) = "C:/Program Files/Ansys Inc/v90/icemcfd/5.1-win"

Note also that the curly brackets (**{...}**) are removed in the original command in order to do variable substitution within, and the square brackets, **[...]**, will do the grouping.

Creating Your Own Procedures (Functions)

By using the **proc** command, you can put the earlier script in a procedure and use the file names as the arguments, so different file names can easily be typed in without re-sourcing the file or restarting Ansys ICEM CFD. The syntax for the **proc** command is:

proc *name args body*

An example procedure using the earlier script might look like this:

```
set wdir D:/user/support/test
proc merge_tetins {infiles outfile} {
global wdir
foreach tetin $infiles {
ic_load_tetin [file join $wdir $tetin]
}
ic_save_tetin [file join $wdir $outfile] 0 0
}
```

Note the **global** command inside the procedure. Any variables defined outside of a procedure are global variables. Any variable set or used inside a procedure, including the arguments, are local variables, and will be unset when the procedure is finished executing. **wdir** is a global variable since it was defined outside a procedure. To access it inside a procedure, it must be declared inside the procedure as **global**.

After reading the script into Ansys ICEM CFD using any of the 4 methods, the procedure will only be defined. It will not be executed. To execute the procedure, type the following in the Ansys ICEM CFD message window or in the shell (in batch mode):

merge_tetins {tetin1.tin tetin2.tin tetin3.tin} new_tetin.tin

You can re-type this at any time with different argument inputs to re-execute the procedure. If you would like to execute the procedure when the script is read, you can add the call of the procedure to the end of the script. A script can have any number of procedures, and any of the procedures can be called from inside the script.

Mathematical Expressions

Mathematical expressions cannot be evaluated in **Tcl** simply be typing them. They must be contained within the **expr** command. For example:

```
    %expr {5+2/7}

5
%expr {5+2/7.0}
5.28571428571
```

The **expr** command does its own substitution, so if you use **"..."** or no grouping at all instead of **{...}**, it will substitute once before the command is executed, and then once again when it is executed. This may not affect the result, but it will be quicker to execute if you only allow one substitution by using the curly brackets.

The order of evaluation is the same as conventional mathematics. 2/7 is done before 5 is added. The resulting type (floating, integer, etc.) will be integer if all of the operands are integer. If any one operand is a floating point, then the result will be a floating point.

Note:

The **expr** command is done on the test conditions for the **if**, **while**, and **for** commands, so you can use mathematical expressions inside without using the **expr** command.

Scripting Commands

These commands are used in MED for writing scripts. All script commands begin with **ic**, for Ansys ICEM CFD.

File Functions

ic_argv

These functions do various high-level and miscellaneous things. Returns the list of command line arguments that were passed after **-batch** and **-script**.

ic_save_project_file *file data* [""] *AppName* [""]

Saves project data to a file.

ic_unload_mesh *quiet* [0]

Unloads the current mesh. This does not save the current mesh if it has not been already saved.

ic_read_external *backend files create* [1] *prec* [1] *tstep* [0] *args*

General resultlib read function. Reads the files specified by the file list using the reader specified by the *backend*. If *create* is 1, the unstructured/structured meshes will be recreated.

ic_set_max_map_size *what size* [""]

Sets the maximum memory size used for CAD or mesh, in bytes. If *size* is "", it returns the current value.

ic_set_program_paths *vals*

Sets the current values for alternate path names for subprograms that are run by Ansys ICEM CFD. The *vals* argument is a list of name/file pairs. The names are the base names of the subprograms, such as *tetra*. The files can be either relative or absolute and need not include the `.exe` in the case of Windows.

ic_set_clean_up_tmp_files *on*

Sets the *remove temp files* flag. If this is 0 then various .tmp files will not get cleaned up. This can help with debugging.

ic_domain_types *args*

Identify what types these domain files are. Returns a list of *struct*, *unstruct*, or *bad*.

ic_convert_struct_to_unstruct *domains_dir numbers topofile famtopo bocofile unsfile options* [""] *prefix* [""]

Converts a structured set of domains to unstructured.

ic_convert_domains_to_unstruct *domains_dir numbers* [""] *unsfile* [""]

Converts raw domains (no topology) into an unstructured mesh.

ic_convert_struct_to_super *domains_dir numbers topofile singledomain listfile pref* [""]

Converts a structured multi-block mesh to single-block.

ic_file_is_ascii *file*

Returns 1 if file is ascii, 0 if not. An error message is returned if the file does not exist, is a directory, or does not have read permissions by the current user.

Stand-alone scripting

A stand-alone scripting interface to ICEM CFD Hexa blocking is available for advanced users. With these tools you should be able to develop vertical or embedded applications with blocking, creating a rich set of APIs or Add-ins to interact with a native blocking object. A README.txt file located in your installation directory under `../icemcfd/win64_amd/icemcfd/blocking-interfaces` contains an introduction to the harness system.

Currently, you can create scripts using Python or C#. The `blocking-interfaces` folder contains two appropriately named subfolders which contain additional details necessary to create scripts plus working examples for these two scripting options. Input data for the example scripts are also included in the `blocking-interfaces` folder.

The `doc/html` folder contains reference information for the interface functions available. The start page at the top of the hierarchy is `index.html`. Many functions are available to query the ICEM CFD data file, and to perform input and output functions. Of particular interest is **`ProcessHexaCommand()`**

which allows you to send ICEM CFD commands (as described in this Programmer’s Guide) to the ICEM CFD session.

Form Creation Functions

Using the form creation functions, it is possible to define forms at a higher level than basic **Tk** widgets. You can create a **form**, which is a top level widget, using **form_init** and then fill it in using other **form_** functions. Most of the functions take the following arguments and not the individual function definition.

- **w**: the widget name for the widget being created. This must be a descendant of the frame that is returned by **form_init**
- **tabdata**: this is the row/column specification that is used with the grid manager. The tabdata string is of the form **row column [options ...]** where row and column are non-negative integers and the options can be:
  - **-cols numcols**: span this many columns
  - **-rows numrows**: span this many rows
  - **-sticky nwes**: some subset of north, west, east, south, to determine the widget alignment
- **activedata**: this is a list of *varname value* ... lists that determine when the widget is active or disabled. A trace is put on each *varname*, and whenever the value changes it is checked to see if it matches a value in the list. If all the *varnames* are okay then the widget is enabled, otherwise it is disabled and grayed out. The values can be either a regular value, or !value which matches anything but the given value. If no values are given then a non-zero value is matched.
- **borderwidth**: the width of the widget border in pixels.
- **relief**: the relief of the widget - should be one of **flat, raised, sunken, ridge, groove**

form_init *w title class* [""] *buttons* [""] *retcmd* [""] *map* [1] *orien* [1] *resize* [""] *dismiss_cmd* [""]

Creates a new top level form, or maps the window if it has already been created. The arguments are:

- **w**: this is the name of the top level form to create
- **title**: the window manager title for the window
- **class**: if non-blank, unmap all windows with this same class name
- **buttons**: a list of button specifications for the row of gray buttons at the bottom. Each spec may be a list with a title and a command, or one of the following standard names:
  - **accept**: closes the form and returns 1 from **form_wait**
  - **reset**: resets all variables back to their original values
  - **cancel**: closes the form, resets the variables, and returns 0 from **form_wait**

  - **done**: same as **accept** except for the button label
  - **dismiss**: same as **accept** except for the button label
  - **close**: same as **accept** except for the button label
  - **nl**: goes to a new row of buttons
- **retcmd**: the command to run when you press return in some field where it is not otherwise bound. The default is to activate the first button at the bottom
- **map**: if 1, maps the window, otherwise not. If 2 then maps the window and prevents it being a transient window
- **orien**: if 1 then the window will be oriented vertically, if 0 horizontally

The return value of **form_init** is a frame widget where all further **form_widgets** for this form should be placed. If it is an empty string it implies that the form has already been created and you should not add any more widgets.

form_side_figure *w var pref* [""] *opts* [""]

Creates an area on the side of the main form that contains a figure which is updated as needed depending on the variable. If no options are given, specify a file name instead of the variable.

form_cancel_class *class except* [""]

Dismisses all forms with the given class except for the one named *except*, if specified.

form_cancel_all *except*

Dismisses all forms.

form_check_visible_class *class*

Returns the forms in the class that are visible.

form_wait *w use_grab* [0]

After the form creation has been completed using **form_finish**, this function can be called which will do a **tkwait** and return only when the form has been dismissed. It returns 1 if the **Accept** button has been pressed and 0 if **Cancel** has been pressed. If *use_grab* is specified as 1, then **proc** will prevent interaction with any other widget until interaction with this form is complete.

form_done *w flag* [1]

Makes the form go away (withdraws it).

form_kill *w*

Destroys the form if it has been created. The next time **form_init** is called, it will return a sub-window name so the form can be recreated from scratch.

form_finish *geo* [-0-0] *map* [1] *autoraise* [0]

This function should be called when the sub-windows have all been created, and the form is ready to map. The arguments are:

- *geo*: the geometry in **wm geometry** format. Also **bot** means the lower-left hand corner and **top** is the upper right-hand corner.
- *map*: if 1, then the window is to be mapped. If 0 the window is to be unmapped.
- *autoraise* : if 1, then any time this window goes behind another it will try to jump to the front. Precautions are taken to prevent fighting between two autoraise forms.

form_colwidth *w num width units* [""]

Makes the width of the given column in the form as specified for *width*. If *units* is non-empty then the width will be taken to be character widths, else it will be considered pixels.

form_rowheight *w num height*

Makes the height of the given row in the form as specified by *height*.

form_canvas *w geo relief borderwidth tabdata xs* [1] *ys* [1]

Creates a canvas widget.

- *geo* is of the form *width* × *height* and gives the size of the canvas.
- If *xs* or *ys* is 1 then an X or Y scroll bar, respectively, is created.

form_table *w coltitles variable relief borderwidth cellw tabdata xs* [1] *ys* [1] *nrows* [1]

Creates a table widget.

form_listbox *w title items click_cmd dclick_cmd geo tabdata activedata* [""] *call_with_index* [0] *d_call_with_index* [0] *multi_select* [0] *xscroll* [0]

Creates a listbox widget. The arguments are:

- *title*: a string to put at the top of the listbox
- *items*: a list of items which go in the listbox initially
- *click_cmd*: if non-empty, a command which is bound to a single left mouse click in the listbox. The item is appended to the command unless *call_with_index* is set in which case the index of the item is used instead.
- *dclick_cmd*: if non-empty, a command which is bound to a double left mouse click in the listbox. The item is appended to the command unless *d_call_with_index* is set in which case the index of the item is used instead.
- *geo*: the value is of the form *width* × *height* and gives the size of the listbox
- *multi_select*: if this is 1, it will be possible to select more than one item at a time, and the *click_cmd* and *dclick_cmd* functions will be called with a list of items rather than just one.

form_listbox_add *w item*

Adds the item to the end of the listbox, unless it is already there.

form_listbox_add_dup *w item*

Adds the item to the end of the listbox, irrespective of whether it is there or not.

form_listbox_delete *w item*

Deletes the item from the listbox. This returns the item right after the deleted one or the one before it, if it was the last.

form_listbox_delete_index *w pos*

Deletes the item at the specified position in the listbox.

form_listbox_change *w old new*

Changes the item *old* to *new*.

form_listbox_select *w item glob* [0]

Makes the given item the selected one.

form_listbox_select_index *w pos*

Makes the item at the given position the selected one.

form_listbox_get_select_index *w*

Returns the index of the selected item.

form_listbox_get_selected *w*

Returns the value of the selected item.

form_listbox_set_selected *w vals*

Sets the value of the selected item.

form_listbox_clear *w*

Deletes all items from the listbox.

form_listbox_set *w items resize* [0]

Sets the items in the listbox to the ones in the list. If *resize* is 1, then make sure the listbox is wide enough to hold all the items.

form_listbox_set_index *w num item*

Sets the item at the given position to be **item**.

form_listbox_get *w num* [""]

Gets the item at the given position, or all items if blank.

form_listbox_size *w*

Returns the number of entries in the listbox.

form_listbox_scale *w size*

Resizes the listbox to have the given number of rows.

form_entry *w var datatype tabdata activedata* [""] *width* [8] *rjust* [""] *save_var* [""] *cmd* [""]

Creates an entry widget. The arguments are:

- *var*: the variable to bind to the widget value
- *datatype*: the data type to check the value against, if non-blank. This may be one of the following types:
  - **float**: a floating point number
  - **float_expr**: a floating point number or **Tcl** expression
  - **float_var**: a floating point number or $var string
  - **float_blank**: a floating point number or blank
  - **float_cmp**: a floating point number greater than 1
  - **int**: an integer
  - **int_blank**: an integer or blank
  - **pct**: a float between 0 and 100
  - **coords**: 3 floats separated by spaces
  - **coords_float**: 3 floats separated by spaces or 1 float
  - **choice**: this keyword is followed by a list of options. The value in the entry must be one of the options
  - **choice_others**: the same as choice but no verification is done
  - **string**: any string
- *width* : the width of the widget

form_text *w title var tabdata activedata* [""] *geo* [40x10] *xscroll* [0] *font* [""]

Creates a text widget. The arguments are:

- **title**: a string to put above the window
- **var**: the variable to bind to the widget's value

- **geo**: the value is of the form *width* × *height* and gives the size of the text area

---

Note:

Unlike the entry widget, the value in the text area is only synchronized when the form is mapped and accepted.

---

form_text_get *w*

Returns the text in a text widget.

form_text_set *w value*

Sets the text in a text widget.

form_text_append *w value*

Appends to a text widget.

form_make_table *rows1*

Formats a table as text.

form_button *w title command tabdata activedata* [""]

Creates a button.

- *title*: the string on the widget
- *command*: the command to bind to the button

form_button_bitmap *w bmap command tabdata activedata* [""]

Creates a bitmap button.

- *bmap*: the bitmap to be drawn on the button
- *command*: the command to bind to the button

form_checkbutton *w title var tabdata activedata* [""] *command* [""] *nonbool* [""]

Creates a check button.

- *title*: the string on the button
- *var*: the boolean variable to assign to the button
- *command*: the command to bind to the button
- *nonbool*: (optional) accept non-boolean variables (e.g. catia v4 import) -fifi

This has special code to deal with parameters.

form_radiobutton *w title var val tabdata activedata* [""] *command* [""] *allow_params* [1]

Creates a radio button.

- *title*: the string on the button
- *var*: the variable to assign to the button
- *val*: the value to assign to the variable when the button is selected
- *command*: the command to bind to the button

This has special code to deal with parameters.

form_menubutton *w title menu tabdata activedata* [""]

Creates a menu button.

- *title*: the string on the button
- *menu*: the menu to bind to the button

form_menubutton_regular *w title menu tabdata activedata* [""]

Creates a menu button, but makes it look like a regular button.

- *title*: the string on the button
- *menu*: the menu to bind to the button

form_optionmenu *w var opts tabdata activedata* [""] *changecmd* [""]

Creates a menu button and menu and defines the text of the button to be the current value of the variable controlled by the menu.

- *var*: the variable whose value is controlled by the menu
- *opts*: a list of {value name} pairs which are added to the menu

form_optionmenu_dynamic *w var func tabdata activedata* [""]

Creates a menu button and menu and defines the text of the button to be the current value of the variable controlled by the menu. The list of available options is returned by a function that is called every time the menu is posted.

- *var*: the variable whose value is controlled by the menu
- *opts*: a list of {value name} pairs which are added to the menu

form_commandmenu *w text commands tabdata activedata* [""]

Creates a menu button and menu and fills it with the given commands.

form_label *w title tabdata activedata* [""] *try_ltext* [1] *justify* [""]

Creates a label with the given value. If *try_ltext* is 1 then the language-specific lookup tables will be checked.

form_label_bitmap *w bitmap tabdata activedata* [""]

Creates a label with a bitmap.

form_label_image *w file tabdata activedata* [""]

Creates a label with an image.

form_label_variable *w var tabdata activedata* [""]

Creates a label attached to a variable.

form_message *w text tabdata activedata* [""]

Creates a message widget.

- *text*: the text to put in the widget

form_scale *w var command orient min max incr tabdata activedata* [""] *on_release* [""] *width* [""]

Creates a scale widget.

- *var*: the variable whose value is controlled by the scale
- *command*: a command to issue each time the scale moves
- *orient*: *v* for vertical and *h* for horizontal
- *min*: the minimum value for the scale
- *max*: the maximum value for the scale
- *incr*: the minimum increment in the value
- *on_release*: a command to be executed when the scale button is released

form_frame *w relief borderwidth tabdata*

Creates an empty frame.

form_label_frame *w label tabdata act* [""]

Creates an empty frame with a label and return the inner part.

form_choiceframe *w*

Creates an empty frame that can be used with the **form_setup_switch** function.

form_underline *w tabdata*

Creates an underline.

form_setup_switch *frame varname opts* [""] *cmd* [""] *change_size* [1]

After creating a number of sub-frames using **form_choiceframe**, which must all be children of a single parent frame, this function sets up a trace on a variable which makes the subframe visible depend on the value of the variable. If the subframes are **$d.sub.val1**, **$d.sub.val2**, etc, and there is a variable **choicevar** which is maybe controlled by a set of radiobuttons elsewhere on the form, and can have the values **val1**, **val2**, etc, then **form_setup_switch $d.sub choicevar** would set this up. The other arguments are:

- *opts*: extra options that can be passed through to the place command to put the subframe in the parent
- *cmd*: a command that is executed after the switch is done

form_scroller *w maxh tabdata hfixed* [0] *wfixed* [0]

Creates a scrolled window.

- *maxh*: the maximum height for the outside window.
- *hfixed*: if non zero, it is the fixed height for the outside window.
- *wfixed*: if non zero, it is the fixed width for the outside window.

The return value is a sub-frame that you can then use to put other widgets in.

form_scroller_adjust *d*

Due to some strange window system interactions, after calling **form_finish** you need to call this function for each scroller in the form, else the size will not be quite right.

External Commands

These commands are used to run external programs or perform OS-dependent things. Unless otherwise noted, they behave similarly to the corresponding UNIX commands. All commands begin with **cmd_**.

cmd_rm *args*

Removes the specified files.

cmd_rmdir *args*

Removes the specified directories.

cmd_rm_r *args*

Removes the specified files or directories.

cmd_cp *args*

Copies the first files into the last named directory, or renames a file.

cmd_cp_r *args*

Recursively copies the first files into the last named directory.

cmd_mv *args*

Moves the first files into the last named directory.

cmd_rename *old new*

Renames *old* to *new*.

cmd_mkdir *dir*

Makes a new directory with the specified name.

cmd_date

Returns the current date.

cmd_chmod *mod file*

Changes the file permissions to the given octal *mod*.

cmd_kill *pid sig* [15]

Sends a signal to the process. On UNIX systems, the *sig* argument gives the signal number.

cmd_uname_n

Returns the current computer's name.

cmd_whoami

Returns the current user name.

cmd_uname_a

Returns information about the current OS.

cmd_tempfile *fn suff* [""]

Returns a file that begins with *fn* that does not currently exist.

cmd_gzip *args*

Runs gzip with the given arguments.

cmd_cat_onto *f1 f2*

Appends *f1* onto the end of *f2*.

cmd_tar *args*

Runs tar with the given arguments.

cmd_uuencode *f u f2* [""]

Runs **uuencode** with the given arguments.

cmd_uudecode uu

Runs **uudecode** on the given file.

cmd_grep *args*

Runs **grep** with the given arguments.

cmd_renice *pri pid*

Changes the priority of process *pid* to *pri*.

cmd_wc *file*

Runs **wc** on the specified file.

cmd_xterm *args*

Opens an Xterm or a Windows command prompt in the current directory.

cmd_ln_s *from to*

Either symbolically link *from* to *to*, or on Windows system, copy it.

cmd_pagesize

Returns the current page size on the system.

cmd_freemem

Returns the amount of free memory, if possible.

cmd_edit_file *file* [""] *back* [0]

Edits the file using the default text editor. If *back* is 1 then return immediately, else, wait until the editor finishes. The return value is the file that was selected (if the original file was blank).

cmd_print_file *filename*

Prints the file on the default printer.

cmd_check_process_exists *pid*

Returns 1 if the process exists on the current machine, else returns 0.

Translation Functions

These commands are used to translate data files from one recognized format to another.

ic_trans_ddn_tetin *ddn tetin*

Transfers the given DDN part file into the named tetin file.

ic_trans_ug_tetin *prt tetin mode* [ug_default] *usr_script* [""] *usr_opts* [""]

Transfers the given Unigraphics part file into the named tetin file.

prt	path to UG part
tetin	path to new Tetin name
mode	translation mode • *ug_default* -- same output as interactive interface • *named* -- output only named entities • *levels* -- output all entities; levels mapped to families • *user* -- run user supplied script
usr_script	custom script supplied by user
usr_opts	options supplied for user script

For example usage, refer to `ic_trans_ug_tetin.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_trans_gems_tetin *prt tetin* [""] *tol* [0.0001] *isRel* [1]

Transfers the given GEMS surface file into the named tetin file.

prt	the GEMS file
tetin	the output file, defaults to a filename derived from the gems filename, prt.
tol	specifies the tetin triangulation tolerance. defaults to 0.0001
isRel	when set, the triangulation tolerance is scaled by the part radius, defaults to 1.

For example usage, refer to `ic_trans_gems_tetin.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_trans_tetin_ddn *tetin iparta pname* [""] *units* [-mm]

Transfers the given tetin file into the named DDN part file.

tetin	tetin file
iparta	DDN IPARTA file
pname	DDN Part Name
units	units flag

Notes:

- If the *pname* argument is given, that is used for the part name. Otherwise the name of the tetin file is used.
- The *units* flag may take on the following values:

-mm (default)	millimeters
-inches	inches
-feet	inches/feet

- DDN likes IPARTA files to reside in a directory called *parts*.
- For example usage, refer to `ic_trans_tetin_ddn.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_trans_tetin_brepxml *tetin brepxmlfile light_export* [""]

Transfers the given tetin file into the named DesignSpace BrepXML file. If the *brepxmlfile* argument is given that is used for the part name. Else the name of the tetin file is used. If *light_export* is specified, then the graphics only BrepXML file is written.

ic_trans_iges_ddn *igesfile ddnpart ddnfile dir_file* [default] *dir_file_user* [""] *data_red* [0] *listing_file* [""] *run_icempt* [0] *trans_dos* [0]

Transfers the given IGES file *igesfile* into the DDN file *ddnpart*. The type of directive file *dir_file* may be one of **none**, **default**, **catia** or **user**, among which **default** is the default; it if is **user** then must be specified. You may also specify data reduction *data_red*, a listing file *listing_file*, the option to run **icemptrun_icempt** after the translation, and the option to translate DOS text *trans_dos*.

ic_trans_iges_tetin *igesfile tetinfile dir_file* [default] *dir_file_user* [""] *data_red* [0] *list_file* [""] *run_icempt* [0] *trans_dos* [0]

Transfers the given IGES file *igesfile* into the tetin file *tetinfile*. The type of directive file *dir_file* may be one of **none**, **default**, **catia** or **user**, among which **default** is the default; it if is **user** then must be specified. You may also specify data reduction *data_red*, a listing file *listing_file*, the option to run **icempt***run_icempt* after the translation, and the option to translate DOS text *trans_dos*.

ic_trans_ddn_iges *ddn_file iges_name listing_file* [""] *dir_file* [""]

Transfers the given DDN part file *ddn_file* to the IGES file *iges_name*. The listing file *listing_file* will be saved if specified, and the directive file *dir_file* if specified.

ic_trans_ddn_tvda *infile outfile selected tvda_dir* [""] *tvda_list* [""]

Transfers the given DDN part *infile* to the TVDA file *outfile*. The DDN selected partname **selected** is specified for the directive file created. The directive file *tvda_dir* is optional, as is the output list file *tvda_list*.

ic_trans_tvda_ddn *infile outfile tvda_dir* [""] *tvda_list* [""]

Transfers the given TVDA file *infile* to the DDN part file *outfile*. The directive file *tvda_dir* is optional, as is the output list file *tvda_list*.

ic_trans_ddn_dxf *file outfile dxf_list* [""] *directive_file* [""]

Transfers the DDN part file *file* to the DXF file *outfile*. The DXF listing file *dxf_list* may be specified and the directive file *directive_file* may also be specified.

ic_trans_dxf_ddn *infile file dxf_list* [""] *directive_file* [""]

Transfer the DXF file *infile* to the DDN part file *file*. The DXF listing file *dxf_list* may be specified and the directive file *directive_file* may also be specified.

ic_trans_idi_tetin *prt tetin* [""] *tol* [0.0001] *isRel* [1] *idi_options* [""]

Transfers the given IDI surface file into the named tetin file.

prt	the IDI file
tetin	the output file, defaults to a filename derived from the idi filename, prt.

For example usage, refer to `ic_trans_idi_tetin.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_trans_proe_tetin_protcl *prt tetin* [""]

Transfers the given Creo Parametric part file into the named tetin file.

prt	the Creo Parametric file
tetin	the output file, defaults to a filename derived from the Creo Parametric filename, prt.

For example usage, refer to `ic_trans_proe_tetin.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

Automatically uses the part file to generate the tetin file.

ic_trans_read_esf_file *prt do_tri* [0] *fam* [""]

Imports an External Scan File.

prt	ESF file to read

fam	name of family containing created surfaces
return	list of created surfaces

Notes:

- If the *fam* parameter is omitted, a family will be created for each surface.
- For example usage, refer to `ic_trans_read_esf_file.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_trans_acis_tetin *acisfile outfile*

Imports an ACIS SAT File.

acisfile	the SAT file
outfile	the tetin output file

Note:

For example usage, refer to `ic_trans_acis_tetin.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_trans_dwg_tetin *dwgfile outfile*

Imports a DWG file.

dwgfile	the DWG file
outfile	the tetin output file

Note:

For example usage, refer to `ic_trans_dwg_tetin.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_trans_anf_tetin *anffile outfile*

Imports an ANF file.

anffile	the ANF file

outfile	the tetin output file

Note:

For example usage, refer to `ic_trans_anf_tetin.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_trans_step_tetin *infile outfile facet* [0] *tolarg* [""] *stepver* [0] *args* [""]

Import STEP or IGES file.

infile	a STEP or IGES file
outfile	the TETIN output file
facet	== 1 - Import as faceted geometry == 0 - Import as spline geometry

Note:

For example usage, refer to `ic_trans_step_tetin.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_trans_tetin_step *infile outfile tolarg* [""]

Exports STEP or IGES file.

infile	the tetin file
outfile	the STEP or IGES file

Notes:

- The type of output (i.e. step or iges) is determined by the extension (i.e. .stp or .igs) of the output file

ic_trans_ps_tetin *infile outfile units* [meter]

Imports a Parasolid file.

infile	a Parasolid file
outfile	the tetin output file

Note:

For example usage, refer to `ic_trans_ps_tetin.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_trans_tetin_ps *infile outfile*

Exports a Parasolid file.

infile	the tetin file
outfile	the Parasolid file

Note:

For example usage, refer to `ic_trans_tetin_ps.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_wb_set_default_properties

Import an Ansys Workbench model.

ic_wb_set_parameters

wb_import_geom [""] *wb_import_mesh* [""] *wb_import_mat_points* [""] *wb_NS_to_subset* [""] *wb_NS_to_entity_parts* [""] *wb_import_solid_bodies* [""] *wb_import_surface_bodies* [""] *wb_import_line_bodies* [""] *wb_import_cad_att_trans* [""] *wb_import_cad_att_pre* [""] *wb_import_sel_proc* [""] *wb_import_sel_pre* [""] *wb_import_mix_res* [""] *wb_import_en_sym_proc* [""] *wb_import_scale_geo* [Default] *wb_import_work_points* [""] *wb_import_delete_solids* [""] *wb_import_create_solids* [""] *wb_import_save_pmdb* [""] *wb_import_load_pmdb* [""] *wb_import_refresh_pmdb* [""] *wb_import_cad_associativity* [""] *wb_import_save_partfile* [""] *wb_import_pluginname* [""] *wb_NS_only* [""] *wb_import_tritol* [0.001] *wb_import_reference_key* [0] *wb_import_lcs* [0]

Set the parameters for the Ansys Workbench reader file import.

ic_wb_brep_read *prt make_absolute_paths* [1] *merge* [0] *mesh* [0] *subset* [0] *geom* [1] *executable* [0] *batch* [0] *refresh_pmdb* [0] *save_pmdb* [""] *entity_parts* [0] *line* [0] *ns_only* [0] *create_material* [0]

Runs the Ansys Workbench reader file import.

ic_wb_brep_load *load_file* [""] *module_name* [""] *doUpdate* [0] *assemId* [0xFFFFFFFF]

Imports a Ansys Workbench model. Returns 1 if build topology is required as a postprocessing step. Returns 0 otherwise.

ic_wb_brep_attach *load_file* [""]

Attach to an Ansys Workbench model.

ic_trans_cos_ccl *file*

Saves the coordinate system to a CFX Command Language (CCL) file.

Display Functions

ic_view *what*

Sets the current viewing position. *what* may be *home*, which sets the home position.

ic_visible *type what on*

Enable/disable visibility for different objects. The *on* argument should be 0 or 1. *type* is one of *geometry*, *unstruct*, or *family*. In the case of *geometry*, *what* should be one of *surface*, *curve*, *point*, *material*, *density*, or *loop*. In the case of *unstruct*, it should be one of the element type names like **TETRA_4**. In the case of *family*, it should be a family name.

ic_set_family_color_for_name *name color*

Sets the color to use for drawing objects in the named family. This applies to geometry and mesh.

ic_display_update *mode* [""] *new_fams* [1] *types* [""]

Updates the display in the GUI. *mode* is one of: *all*, *unstruct*, *struct*, *geom*, **""** implies *all*. *new_fams* determines whether or not to update the family list in the GUI. *types* allows you to make visible entity types appropriate to the mode (For example, {TRI_3} for unstruct).

Geometry Functions

ic_load_tetin *filenames tri_tolerance* [0] *keep_model_params* [1] *blanks* [0] *quiet* [0]

These functions manipulate the geometric data which is loaded in from the **tetin** file. Geometric entities are referenced by type and name. For every entity of a given type there is a unique name, and every entity is in a given family which is generally transferred to the mesh that is created from the geometry. The types are:

- **surface**: surfaces, which may be either trimmed NURBS surfaces or faceted (from STL or other triangular formats). Surfaces correspond to regions of triangles or quads in the mesh, and nodes in the mesh that fall on a surface have a dimension of 2.

- **curve**: either NURBS curves or piecewise linear paths. Curves correspond to bar elements and nodes with dimension 1.

- **point**: prescribed points. These correspond to NODE elements in the mesh and nodes with dimension 0.

- **material**: material points. These define connected regions, where all volume elements are assigned the family of the material point.

- **loop**: a list of curves which defines a closed loop, which is used by the **Quad** surface mesher.

- **density**: a density polygon is a set of points that define a convex hull, in which the size of the tetra elements must be no greater than a specified value.

Load the specified tetin files. The full path names must be given. If the *tri_tolerance* is specified, that becomes the new triangulation tolerance for the file, overriding what is specified.

Notes:

- The tetin file will be merged with any existing geometry already loaded. See also **ic_unload_tetin** (p. 36).

- On Windows use the "/" character as the path delimiter instead of the backslash "\" character. For example:

```
ic_load_tetin c:/speed_racer/mach5.tin
```

ic_empty_tetin

Creates an empty geometry database.

ic_save_tetin *file only_visible* [0] *v4* [0] *only_fams* [""] *only_ents* [""] *v10* [0] *quiet* [0] *clear_undo* [1]

Saves the current geometry data to the given file name. The full path name must be specified. If *only_visible* is 1 then only the visible data will be saved. If *v4* is 1 then the tetin file will be saved in Ansys ICEM CFD 4.x compatibility mode.

Note:

On Windows use the "/" character as the path delimiter instead of the backslash "\" character. For example:

```
ic_save_tetin c:/speed_racer/mach5.tin
```

ic_unload_tetin *quiet* [0]

Unloads the current geometry data.

ic_geo_import_mesh *domains* [""] *do_seg* [1] *no_orfn* [1] *do_merge* [1]

Imports a mesh file and creates a tetin database. Surface elements are turned into surfaces (by family), bar elements to curves, node elements to prescribed points, and one volume element per family to material points. If domains is "" then it will be imported from the loaded unstructured mesh. If *do_seg* is 1, then the curves and surfaces will be segmented by connectivity, else they will be kept as one object per family. If *no_orfn* is 1, then any elements that were in the **`ORFN (0)`** family will be moved to a new family called **`MOVED_ORFN`**.

update_surface_display *obj*

Utility to update display of surface and related edges, vertices.

ic_geo_export_to_mesh *merge fams* [""] *quiet* [0]

Copies the current geometry into the mesh database. This is the opposite of **ic_geo_import_mesh**. Surfaces will be turned into families of triangles, etc. If *merge* is 1, then all surfaces, etc will keep their families, and if 0, then the geometric objects will each be mapped to a separate family in the mesh.

ic_ddn_app *type partname partdir extra_cmds batch* [1]

Runs either DDN or a DDN GPL application.

- *type*:

  type may be **ddn**, **mif**, **input**, or **frontal**.

- *partname* and *partdir*:

  The *partname* and *partdir* arguments map to the DDN *pn* and *db* command line arguments.

- *extra_cmds*:

  extra_cmds is a list of lines that will go into the input file after the commands to start up the GPL application.

- *batch*

If *batch* is 1 (the default) then DDN will be run without graphics.

Note:

This command is currently broken in MED 4.2 08/16/01

ic_list_ddn_parts *dir update*

Updates the DDDN directory file.

dir	name of parts directory
update	1 = update the directory
return	List of DDN parts

ic_geo_summary

Prints a summary of the geometry in the message window.

ic_geo_lookup_family_pid *fam*

Returns the internal numeric id (*pid*) of the family. This is not a safe function to use in general because the *pid* is not guaranteed to stay the same between one invocation of the program and the next, and no scripting commands use it as an argument.

ic_geo_is_loaded

Reports if any geometry is loaded.

ic_geo_is_modified

Reports if the current geometry has been modified since the last save.

ic_geo_valid_name *name no_colon*

Changes a name into a valid family/entity name.

ic_geo_set_modified *on*

Sets the modified flag for the current geometry. This should not be used for most operations since they set this flag themselves.

ic_geo_check_family *name*

Checks whether a family exists.

ic_geo_check_part *name*

Checks whether a part exists.

ic_geo_new_family *name do_update* [1]

Creates a new family if it is not already there. Returns 1 if a new family was created, or 0 if it already existed.

Note:

The newly created family will not appear in the interactive family list unless you issue the **update_family_display** command interactively.

ic_geo_new_name *type prefix*

Creates a new, unused name for an entity in a family.

ic_geo_get_unused_part *prefix no_first_num* [0]

Creates a new unused part or family name.

ic_geo_delete_family *names*

Deletes a family, or a list of families.

ic_geo_params_blank_done *type reset* [0]

Blanks those entities that have some meshing parameters already defined. If *reset* == 1, then the entities blanked entities are unblanked.

ic_geo_match_name *type pat*

Returns the names of the objects that match a given pattern.

ic_geo_update_visibility *type vis_fams visible*

Changes the visibility so that only objects with the given families and type are visible or not, depending on the *visible* option. If *vis_fams* is `*skip*` then they are retained and the *type* is checked. If a family is not listed in the family list then it is ignored.

ic_geo_get_visibility *type name*

Returns whether an object is visible or not.

ic_geo_set_visible_override *type name val*

Sets or unsets the **visible_override** flag. If this flag is set for an object then it is always visible no matter what types and families are enabled. This is needed for geometry subsets.

ic_geo_temporary_visible *type objects vis force* [0]

Temporarily blanks or unblanks an object. This will not go away when you change anything larger scale. If *objects* is set to all then all will be blanked or unblanked.

ic_geo_get_temporary_invisible *type* [entity]

Gets the temporarily blanked entities. Returns a list of "type name", or an empty list if no entity was blanked. *Type* can be entity (for all types), point, curve, or surface.

Example usage: ic_geo_get_temporary_invisible [type]

ic_geo_set_visible_override_families_and_types *fams types*

This is a helper function that sets the **visible_override** flag for all objects in some families and types, and clears it for all others. Note that after calling this function all the visible flags on the non-override families will be off.

ic_redraw_geometry *type name*

This redraws the geometry in case something changes.

ic_geo_incident *type names even_if_dormant* [0]

Returns what objects of higher dimensionality are incident to this one. Surfaces are incident to curves and curves are incident to points.

ic_geo_surface_get_objects *surface type* [embedded_points]

Returns a list of objects associated to a surface.

ic_geo_loop_get_objects *loop type* [surface]

Returns a list of objects associated to a loop.

ic_geo_surface_edges_incident_to_curve *surfname curvename*

Returns the edges in a triangulated surface which are incident to a given curve.

ic_geo_surface_normals_orient *refsurfname* [""] *outward* [1]

Reorients the normals of surfaces in a model with respect to the given reference surface *refsurfname* in direction given *outward* [0|1]. If there is no material point, *outward* means reverse reference surface before reorienting with respect to it.

ic_geo_get_side_surfaces *tol how list* [""]

Returns a list of surfaces whose normal of magnitude *tol* does not intersect another surface.

ic_geo_boundary *type names* [""] *outer* [0] *even_if_dormant* [0] *embedded* [0]

Returns the objects of lower dimensionality which are the boundary of the one specified. Points are the boundary of curves and curves are the boundary of surfaces and loops. The list returned has a sublist for each element in *names* such that **`{{c00 c01} {c10 c11} ...}`**, unless names is one element, in which case the returned list goes as: **`"c00 c01`**".. *names* may be one name, or a list of names. *outer* may be 0 or 1, indicating what kind of boundaries. *even_if_dormant* may be 0 or 1, indicating which additional boundaries. *embedded* may be 0 or 1, indicating which additional boundaries.

ic_geo_object_visible *type names visible*

Changes the visibility of a specific object or objects. The *visible* argument is 0 or 1.

ic_geo_configure_objects *type simp shade st sp sh sq names wide dnodes count nnum tnum norms thickness lshow lfull lst lsq comp* [0] *grey_scale* [0] *transparent* [0] *count_quad* [0] *dormant* [0] *protect* [0] *hardsize* [0]

Configures the attributes of all the visible objects of a given type. The arguments are:

- *simp* : the level of simplification. 0 shows the full detail of the object, and higher values simplify them more: 5 is very simple. In the case of faceted geometry, this value is 1/20 of the angle that is used to extract internal curves
- *shade* : how to draw the objects: one of wire, solid, solid/wire, or hidden.
- *st* : if 1 show sizes appropriate for Tetra.
- *sp* : if 1 show sizes appropriate for Prism.
- *sh* : if 1 show sizes appropriate for Hexa.
- *sq* : if 1 show sizes appropriate for Quad.
- *names* : if 1 show the names of the objects.
- *wide* : draw wide lines. This value plus 1 is the line thickness.
- *dnodes* : if 1 display the nodes of the object.
- *count* : if 1 show curves in different colors:
  - *green*: no surfaces are adjacent to this curve
  - *yellow*: just one surface is adjacent to this curve
  - *red*: two surfaces are adjacent to this curve
  - *blue*: more than two surfaces are adjacent to this curve
- *nnum* : draw node numbers (internal to the surface or curve, useful for debugging)
- *tnum* : draw triangle or segment numbers (internal to the surface or curve, useful for debugging)
- *norms* : draw normals of non-mesh surfaces
- *comp* : show composite curves
- *grey_scale* : show all grey scaled
- *transparent* : show all transparent
- *count_quad* : show curve element count
- *dormant* : show dormant elements (points/curves)
- *protect* : show protected elements (points/curves)
- *hardsize* : show hard sized curves

ic_geo_configure_one_attribute *type what val*

Configure one attribute of a whole type.

ic_geo_configure_one_object *type name what val* [-]

Configure the attributes of one object.

ic_geo_list_families *only_material* [0] *non_empty* [0]

Lists the current families used by geometry objects. If *only_material* is 1 then limit the listing to families used by materials. If *non_empty* is 1 then list only families which are non empty.

ic_geo_list_parts *prefix* [""] *non_empty* [0]

Lists the current parts.

ic_geo_check_part *name*

Checks if a part exists.

ic_geo_list_families_in_part *part*

Lists the current families in a part.

ic_geo_list_families_with_group *gname*

Lists the families in a group.

ic_geo_list_parts_with_group *gname*

Lists the parts that have some family in a group.

ic_geo_family_is_empty *fam*

Returns whether or not the family is empty of entities.

ic_geo_family_is_empty_except_dormant *fam*

Returns whether or not the family contains only dormant entities.

ic_geo_non_empty_families

Lists all the non-empty families.

Note:

This does not check whether there are directives.

ic_geo_non_empty_families_except_dormant

Lists all the families containing only dormant entities.

Note:

This does not check whether there are directives.

ic_geo_num_objects *type*

Returns the number of objects of the given type.

ic_geo_list_visible_objects *type even_if_dormant* [1]

Returns the number of visible objects of the given type.

ic_geo_num_visible_objects *type any* [0]

Returns the number of visible objects of the given type. If *any* is 1, specify whether that number is more than 0.

ic_geo_num_segments *type name*

Returns the number of triangles or bars, and nodes in this object.

ic_geo_set_family *type newfam how objs rename* [1]

Changes the geometry with the given *type* and name to family *newfam*. The first argument tells the type of geometry objects: surface, curve, material, point, density, or loop. The second argument is the new family name to be set. The third argument tells how to select the objects and the fourth is the list of object specifiers. *how* can be one of the following:

- *names*: a list of the names of the objects
- *numbers*: a list of the internal numbers (not for general consumption)
- *patterns*: a list of glob-patterns that match the names of the objects
- *families*: a list of the families to select
- *all*: all objects of that type (the objects list is ignored)
- *visible*: all visible objects of that type (the objects list is ignored)
- *rename*: if 1, change the name of the object to match the new family, if appropriate. Note that if this function is called as part of a larger script, the renaming might break things if other parts of the code think they know the names of the objects they are dealing with and those names change beneath them.

ic_geo_set_part *type names newpart rename_part* [1]

Moves geometry from one part to a new one. This has to create the new part name and copy the boundary conditions if necessary so that the other groups in the family are not disturbed.

ic_geo_set_name *type name newname make_new* [0] *warn* [1]

Change the geometry with the given type and name to name *newname*. If *make_new* is 1 then an unused name that starts with *newname* is used and this value is returned from the function. If possible *newname* is used without modification. If *make_new* is 0 then any objects of that same type and name that already exist will be deleted first.

ic_geo_rename_family *fam newfam rename_ents* [1]

Rename the family. All objects in *fam* will now be in *nfam*. If *rename_ents* is set, family entities will be renamed.

ic_geo_replace_entity *type e1name e2name*

For two geometry entities of *type*, the first, of name *e1name*, will be replaced by the second, of name *e2name*, as well as being put into the family of the first entity and having the meshing parameters copied from the first to the second. The name of the first entity is appended with _OLD and put into the family ORFN.

ic_geo_get_ref_size

Returns the reference mesh size. This is used to scale all meshing parameter values for display to the user.

ic_set_meshing_params *type num args*

Set or get the meshing parameters associated with the model or the geometry. The *type* and *num* arguments define what the parameters are being defined for. The remaining arguments are name/value pairs, so that the function call might look like

ic_set_meshing_params surface 2 emax 10 erat 13

The *num* argument can also be a name. Any or all of the meshing parameters can be specified, and the ones not given are not modified. Note that all the sizes are in absolute units, not factors of the reference size. This is different from what you see in the GUI. The *type* can be one of the following:

- *global*: set or get the global parameters like natural size, etc. The *num* argument is ignored. The parameters that are accepted are:
  - *gref*: the reference size for the model
  - *gmax*: the maximum size of any element in the mesh
  - *gnat*: the natural size value
  - *gnatref*: the natural size refinement factor
  - *gedgec*: the edge criterion
  - *gcgap*: the number of cells allowed in a gap
  - *gttol*: the triangularization tolerance
  - *gfast*: if the value is 1 then set fast transition
  - *igwall*: if the value is 1 then ignore wall thickness
  - *grat*: the growth ratio value
- *curve*: set or get the parameters on curve *num* The parameters that are accepted are:
  - *emax*: the maximum element size
  - *ehgt*: the maximum height

  - *erat*: the size expansion ratio
  - *ewid*: the number of layers of tetrahedra of the same size that should surround a surface
  - *nlay*: the number of quad offset layers
  - *emin*: the minimum size
  - *edev*: the deviation value
- *surface*: set or get the parameters on surface *num* The parameters that are accepted are:
  - *emax*: the maximum element size
  - *ehgt*: the maximum height
  - *erat*: the size expansion ratio
  - *hrat*: the height expansion ratio
  - *ewid*: the number of layers of tetrahedra of the same size that should surround a surface
  - *nlay*: the number of prism layers
  - *emin*: the minimum size
  - *edev*: the deviation value
- *point*: set or get the parameters on prescribed point *num* The parameters that are accepted are:
  - *ehgt*: the maximum height
  - *erat*: the size expansion ratio
  - *hrat*: the height expansion ratio
- *density*: set or get the parameters on density volume *num* The parameters that are accepted are:
  - *emax*: the maximum element size
- *loop*: set or get the parameters on loop *num* The parameters that are accepted are:
  - *etyp*: the element type
- *curve_fam*, *surface_fam*, *point_fam*: set or get the parameters on all objects of the family *num* (in this case *num* is not a number but a family name). The accepted parameters are the same as the ones listed for individual objects.

The return value of this function is a list of names and values in the same format as the arguments, which are the values after the modification. Therefore to get the current values you can use **ic_set_meshing_params***typenum* with no other arguments.

Note:

If you give a family instead of a number then you will get the parameters only for one of the objects in that family.

ic_get_mesh_growth_ratio

Returns mesh growth ratio.

ic_get_meshing_params *type num*

Returns the meshing parameters. This has the advantage over **ic_set_meshing_params** that it is not recorded in the replay file.

ic_geo_scale_meshing_params *types factor*

Scales the meshing parameters on geometric entities of *types* by a *factor*. If *types* is *all* then rescales entities of types "surface curve point material density loop".

ic_geo_set_curve_bunching *curves args*

Sets curve bunching.

ic_geo_get_curve_bunching *name*

Gets curve bunching.

ic_geo_create_surface_segment *name how args*

Creates new surfaces by segmenting an existing one. The *name* argument is the name of the existing surface. The *how* argument describes how to do the segmentation:

- *angle*: split the surface where the angle exceeds the given value (see below)
- *curve*: split the surface at the named curves (see below)
- *plane*: split the surface at the given plane (see below)
- *connect*: split the surface by connectivity

The remaining optional arguments are:

- *angle*: the angle to use for splitting
- *mintri*: the smallest number of triangles to allow in a surface
- *curves*: a list of curve names to use for splitting
- *plane_p*: the point to define the plane (in the current LCS)

- *plane_n*: the normal to define the plane
- *keep_old*: if 1 then keep the old surface after segmentation. The default is 0

The return value is a list of numbers which are the newly created surfaces.

ic_geo_create_curve_segment *name how args*

Creates new curves by segmenting an existing one. The *name* argument is the name of the existing curve. The *how* argument describes how to do the segmentation:

- *angle*: split the curve where the angle exceeds the given value (see below)
- *plane*: split the curve at the given plane (see below)
- *connect*: split the curve by connectivity

The remaining optional arguments are:

- *angle*: the angle to use for splitting
- *minedge*: the smallest number of edges to allow in a curve
- *point*: point for splitting
- *plane_p*: the point to define the plane
- *plane_n*: the normal to define the plane
- *keep_old*: if 1 then keep the old curve after segmentation. The default is 0

The return value is a list of numbers which are the newly created curves.

ic_geo_split_curve *curve points*

curve	name of the curve
points	list of points
return	names of curve segments

Note:

This splits the curve in the specified order of the points

ic_geo_split_curve_at_point *curve point tol* [0]

Splits a curve at a point.

curve	name of the curve
point	may be a prescribed point name or {xyz} coordinates
tol	(optional) *on curve* tolerance
return	names of curve segments (usually 2)

Notes:

- If the trim operation does not segment the curve (e.g. trim at curve endpoint), the result string will be empty.
- If the function returns with the error status set, the result string will contain an error message.
- For example usage, refer to `ic_geo_split_curve_at_point.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_create_loop *name fam how curves all_separate surfs fams* [""] *pts* [""] *crs* [""]

Creates a loop with the given name and family using the specified curves. If *how* is *names* then *curves* is a list of the names of the curves to use, and if it is *families* then it is a list of family names. If *all_separate* is 1 then all the curves are considered individually and if it is 0 then they are all taken together to create the loop. Surfaces are associated to the loop if a list *surfs* of names of surfaces is given. Optionally the loops can be set to the families in the list *fams* in order, in the case of *all_separate* is 1. Points can be added to a loop if a list *points* of names of points is given.

ic_geo_modify_loop *name curves surfs* [""] *pts* [""] *crs* [""]

Modify 1 loop with the given name using the specified curves, points and surfaces. A list of names of *curves* must be given. Surfaces are associated to the loop if a list *surfs* of names of surfaces is given. Points/corners can be added to a loop if a list *points* of names of points/corners is given.

ic_geo_pick_location *args*

Selects a geometric entity on the screen. Arguments are:

- **line***{{x y z} {x y z}}*: the line in model space
- **type***type*: one of: entity, surface, curve, point, material, loop, or density

The return value is a list that contains the type, name, and location on the object.

ic_geo_get_object_type *type names*

Determines whether an object is type **param** (a trimmed NURBS curve or surface) or **mesh** (a faceted surface or piecewise linear curve). If both types are present in the list of names **mixed** is returned.

ic_geo_trim_surface *surf curves build_topo* [1]

Trims the surface by the curves. This creates a new surface with the name of the old surface followed by **.cut.$n**.

ic_geo_intersect_surfaces *fam surfs bsp_flag* [""] *multi_flag* [""]

Intersects the surfaces and creates new intersection curves in the given family.

family	family containing surface
surfs	list of surfaces
bsp_flag	== "use_bsp" - create curves as b-spline

multi_flag	== "multi_crv" - create multiple curves
return	names of created curves

Notes:

- When *multi_flag* is set to "*multi_crv*", the function will produce 1 curve for each connected component of the intersection. When *multi_flag* is set to "", the function will produce 1 unstructured curve.
- Setting the *bsp_flag* to "*use_bsp*" implies setting the *multi_flag*
- For example usage, refer to `ic_geo_intersect_surfaces.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_intersect_surfs_by_groups *groups fam* [""] *bsp_flag* [""] *multi_flag* [""]

Intersects surface groups.

groups can be a list of two forms:

- {srf1 srf2 srf3 ...}
- {{srf1 srf2 srf3 ...} {srf4 srf5 srf6 ...} {srf7 srf8 srf9 ...} ...}

In the first form, every surface is intersected with each other, but curves created from each surface pair intersection will be separate. In the second form, each set is intersected with every other set. Surfaces within each set will not be intersected with each other. Separate curves are still generated from each surface pair intersection.

ic_geo_create_unstruct_curve_from_points *name fam pts*

Creates a piecewise linear curve from the points. This curve is given the specified name and family. *pts* is a list of triples of floating point numbers or list of prescribed point names and they are connected in order. The points are in the current local coordinate system.

ic_geo_create_unstruct_surface_from_points *name fam pts*

Creates a surface from 4 points, with the given name and family. *pts* is a list of 4 triples of floating point numbers or list prescribed point names and 2 triangles are created to make a rectangular surface. The points are in the current local coordinate system.

ic_geo_create_empty_unstruct_surface *name fam*

Creates an empty surface, with the given name and family.

ic_geo_create_empty_unstruct_curve *name fam*

Creates an empty curve, with the given name and family.

ic_geo_create_curve_ends *names*

Creates points at the ends of the named curve.

names	names of curve

return	names of created points

Note:

This function creates new points at curve endpoints as needed. See also **ic_geo_mod_crv_set_end** (p. 49).

ic_geo_mod_crv_set_end *crv pnt crvend tol* [-1]

Sets the curve end.

crv	name of curve
pnt	name of prescribed point
crvend	curve end indicator
tol	merge tolerance
return	name of curve vertex

Notes:

- The curve end parameter, *crv1end*, takes the values:

  start point 0
  end point 1

- If the curve end is already associated, the function will take no action. The name of the curve end will be returned in any case.
- If the merge tolerance is set negative, the tolerance will default to the triangulation tolerance.
- If the function returns with the error status set, the result string will contain an error message.
- For example usage, refer to `ic_geo_mod_crv_set_end.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_crv_get_end *crv crvend*

Returns the curve end.

crv	name of curve
crvend	curve end indicator
return	name of curve vertex

Notes:

- The curve end parameter, *crv1end*, takes the values:

start point 0
end point 1

- If the curve end is not already associated, the function will return an error which must be caught.

ic_geo_create_points_curveinter *curves tol fam name* [""]

Creates points at the intersection of curves. *curves* is a list of curves to intersect.

ic_geo_create_point_location *fam pt in_lcs* [1]

This function has been replaced by **ic_geo_cre_pnt**

ic_geo_create_material_location *fam pt*

This function has been replaced by **ic_geo_cre_mat**

ic_geo_create_density *name size pts width* [0.0] *ratio* [0.0] *strfac* [1.0] *strvec* [""]

Creates a density polygon from a set of points. It is given the specified name and tetra size. Stretch factor with direction may be specified optionally.

name	name of polygon
size	tetra size (must be > 0)
pts	list of points (must have 4 or more)
width	size of constant spacing layer
ratio	expansion rate
strfac	stretch factor
strvec	stretch direction

Note:

Points may be passed either as 3-tuples or as names of prescribed points.

ic_geo_extract_points *names angle*

Extracts points from curves based on the angle between adjacent segments in degrees.

ic_geo_extract_curves *names bound angle minedge*

Extracts curves from mesh surfaces. If *bound* is 1 then only the boundary of the surface is extracted. If it is 0 then the angle is used to determine feature lines. The *minedge* argument determines what the smallest curve that will be extracted is.

ic_geo_create_surface_edges *names*

Creates curves based on the edges of a surface.

ic_geo_get_srf_edges *srf*

Returns any curves associated as edges to a surface.

ic_geo_stats *type name*

Returns some statistics about the object. This is a readable string that says what the type is, how many triangles, nodes, etc.

ic_geo_get_point_location *name*

Given the name of a point, returns its location.

ic_geo_get_material_location *name*

Given the name of a material point, returns its location.

ic_set_point_location *args*

Sets the location of a point or points. The names and locations must come in pairs.

ic_set_material_location *args*

Sets the location of one or more material points. The names and locations must come in pairs.

ic_delete_material *names*

Deletes a material.

ic_geo_check_objects_exist *type args*

This function checks to make sure the objects exist - it returns the list of names that were found. If no objects with the given type and names were found it returns an empty list.

ic_geo_get_objects *types fams* [""] *even_if_dormant* [0]

This function returns all the objects of the given type and family. If the family does not exist it returns an empty list, and if it is "" then it returns all objects of all families.

ic_geo_count_in_family *types fams*

Returns the number of objects of the given type in the given family.

ic_geo_objects_in_family *types fams*

Returns a list of objects in the given family name.

ic_geo_objects_in_parts *types parts*

Returns a list of objects (type/name pairs) in the given parts.

ic_geo_get_internal_object *type name*

Returns the internal object associated with a name. This is a bit of a back door.

ic_geo_get_name_of_internal_object *obj*

Returns the name of an internal object. This is a bit of a back door.

ic_geo_get_text_point *type name*

Returns the text point list for an object, specified by type and name. The list is: "{xloc yloc zloc} {xdir ydir zdir}"

ic_geo_get_centroid *type name*

Returns the centroid for an object, specified by type and name. The return is: "{xloc yloc zloc}". This function only works for curves at this time.

ic_geo_num_segments *type name*

Returns the number of segments or triangles in the object. This returns 2 numbers - the number of segments and the number of nodes.

ic_geo_num_nodes *type name*

Returns number of nodes in the object.

ic_geo_get_node *type name num*

Returns a node in a mesh curve, surface or density polygon.

ic_geo_drag_nodes *type name ptnums startpts startmouse spos svec how*

Allows you to interactively drag nodes in surfaces and curves.

ic_geo_drag_point *name startpt startmouse spos svec*

Allows you to interactively drag prescribed points.

ic_geo_drag_material *name startpt startmouse spos svec*

Allows you to interactively drag material points.

ic_geo_drag_body *name startpt startmouse spos svec*

Allows you to interactively drag body points.

ic_geo_project_point *type names pt dir* [0 0 0] *tan_ext* [0]

Project a point to a set of objects. *dir* is the vector along which to project, or **0 0 0** if the nearest point is desired. *dir* is used only in the case of surfaces.

type	point, curve, surface
names	list of entity names
pt	point -- either entity name or 3-tuple
dir	projection vector, {0, 0, 0} for nearest point
tan_ext	project to tangential extension (curves only)

ic_geo_project_and_move_point *type names pt dir* [0 0 0] *tan_ext* [0] *fam* [""]

Project and move a point to a set of objects. *dir* is the vector along which to project, or **0 0 0** if the nearest point is desired. *dir* is used only in the case of surfaces.

type	point, curve, surface
names	list of entity names
pt	point -- either entity name or 3-tuple
dir	projection vector, {0, 0, 0} for nearest point
tan_ext	project to tangential extension (curves only)
fam	part name (if blank, use the part name of the point)

ic_geo_project_coords *type names ptloc dir* [0 0 0] *tan_ext* [0]

Project coordinates to a set of objects. *dir* is the vector along which to project, or **0 0 0** if the nearest point is desired. *dir* is used only in the case of surfaces.

ic_geo_nearest_object *type pt dir* [0 0 0] **tol** [0]

Projects a point to a set of objects, and return the name of the best one and the location. *dir* is the vector along which to project, or **0 0 0** if the nearest point is desired. *pt* may either be an XYZ location or the name of a prescribed point. The *tol* argument is used for intersecting a line with curves.

Note:

Curve projection is not yet implemented.

ic_geo_project_curve_to_surface *crvs surfs name fam* [""] *new_name* [0] *bld_topo* [0]

Projects one or more curves to one or more surfaces.

crvs	list of curve names
surfs	list of surface names
name	base name of created curve
fam	family of created curves
new_name	do not reuse existing names
bld_topo	trim surfaces by projected curves
return	name(s) of created curve(s)

ic_geo_create_surface_curves *crv1 crv2 name*

Creates a faceted surface from *crv1* and *crv2*. If the curves are not connected, the new surface will connect straight across the gaps.

name is the name of the new surface.

ic_geo_create_surface_curtain *crvs surfs name fam bld_topo* [0] *quiet* [0]

Creates a curtain surface between one or more curves and a surface.

crvs	list of defining curves
surfs	surface(s) to project to
name	base name of created surfaces
fam	family of created surfaces
bld_topo	run the topology builder
quiet	suppress diagnostic messages

ic_geo_set_node *type name num pt*

Moves a node on an object.

ic_geo_get_family *type name*

Returns the family for an object.

ic_geo_get_part *type name*

Returns the part for an object (just the family with stuff after the first : removed)

ic_build_topo *args*

Builds the topology information from the geometry data. Arguments are:

- *tolerance*: the gap tolerance
- **-angle***angle*: the angle for extraction
- **-filter_points**: extract the cad points
- **-filter_curves**: extract the cad curves
- **-delete_degenerate**: delete degenerate surfaces
- **-save_old**: do not delete existing entities.
- **-quiet**: suppress diagnostic messages
- **-no_reg_surf**: do not trim/regularize surfaces
- **-no_concat**: do join edge curves
- **-create_interior**: create interior curves on mesh surfaces
- **family_name**: name(s) of family(ies) to operate on.

ic_geo_default_topo_tolerance_old

Get a good tolerance for the current geometry.

ic_delete_geometry *type how objects* [""] *report_err* [1] *even_if_dormant* [0]

Deletes geometry objects. The first argument tells what kind of objects to delete: surface, curve, material, point, density, or loop. The second argument tells how to select the objects and the third is the list of object specifiers. *how* can be one of the following:

- *names*: a list of the names of the objects
- *numbers*: a list of the numbers of the objects
- *patterns*: a list of glob-patterns that match the names of the objects
- *families*: a list of the families to select
- *all*: all objects of that type (the objects list is ignored)
- *visible*: all visible objects of that type (the objects list is ignored and can be left out)
- *blanked*: all blanked objects of that type (the objects list is ignored and can be left out)
- *objects*: the internal handles for the objects
- *report_err*: optional argument - if 0 then do not report errors if objects do not exist

ic_geo_pnt_mrg_inc_crv *how objects*

Deletes a point and joins incident curves.

- *how* - defines how points are selected. See **ic_delete_geometry** (p. 54).
- *objects* - the internal handles for the objects

ic_facetize_geometry *type name args*

Makes a new geometric entity that is the faceted or piecewise linear equivalent of the old one. Optional arguments are:

- **noisy***on* : if 1 print status information
- **replace***on* : if 1 replace the old object, otherwise make a new one.

ic_move_geometry *type args*

Moves an existing geometry entity. The *type* argument gives the type: point, curve, surface, material, density, loop, or all. The other arguments are:

- *names*: a list of the names of the objects
- *type_names*: a list of pairs with the types and names of the objects
- *numbers*: a list of the numbers of the objects
- *patterns*: a list of glob-patterns that match the names of the objects (* = all objects)
- *families*: a list of the families to select
- *objects*: the internal handles for the objects

- *cent*: a list of X Y Z giving the center for rotation, scaling, and mirroring, which could be "centroid"
- *translate*: the X Y Z values for translating the entity
- *rotate*: the number of degrees to rotate the object about the axis given by *rotate_axis*
- *rotate_axis*: the vector giving the axis to rotate about
- *scale*: a scale value, or 3 values giving the factor to scale in X Y and Z
- *mirror*: the axis about which to mirror

The translation vector, center of rotation, and rotate axis should be specified in the current local Cartesian coordinate system.

ic_geo_duplicate *type name newname* [""] *facetize* [0]

Duplicates an existing geometrical entity. If the *newname* is given then that is used, otherwise a name is generated automatically. If *facetize* is specified then bspline surfaces are turned into facets.

ic_geo_fixup *mesh* [0]

Fix problems in surfaces and curves like duplicate triangles, unused nodes, etc.

ic_geo_min_edge_length *args*

Returns the minimum edge length on a list of surfaces or curves. Arguments are:

- **surface***val*: list of surface names
- **curve***val*: list of curve names

Example usage: set surfaces "surf1 surf2 surf3" set curves "curv1" ic_geo_min_edge_length surface $surfaces curve $curves

ic_geo_coarsen *args*

Simplify surfaces or curves by coarsening them. The arguments are:

- **tol***val*: the tolerance to use for coarsening, which determines how far from the original surface the new nodes can move (default 0)
- **surfaces***namelist*: the surfaces to coarsen.
- **curves***namelist*: the curves that define prescribed nodes that should not be coarsened (default none)
- **points***namelist*: the points that should not be coarsened away (default none)
- **auto_curves***on* : UNIMPLEMENTED - automatically determine what curves to preserve
- **auto_points***on* : UNIMPLEMENTED - automatically determine what points to preserve
- **n_iter***num*: how many iterations of coarsening to try (default 16)

- **noisy***on* : print status information

ic_geo_gap_repair *args*

Perform geometry repair. Arguments are:

- **toler***val*: the geometry tolerance
- **toler_max***val*: the maximum gap tolerance
- **partial***val*: allow partial repairing if set, default is 0
- **yellow***val*: yellow curves only if set, default is 1
- **do***val*: Close gaps if value is 1, match surfaces if 2, close one hole if 3, and close multiple holes if 4.
- **build_topo***val* : build topology if set, default is 1 (for do = 1 only)
- **new_family***name* : the family name for new geometry
- **new_name***name*: the entity name for new geometry
- **quiet***val* : Quiet operation if set. Default is 0. For do = 1 it selects re-intersection, if 1, fill, if 2, blend, if 3, and Y-closure if 4.
- **db***val*: Do some checking and printing if > 0, print less messages if < 0. Default is 0.
- **curves***names* : the names of the curves to do

Return value is 0 if there was an error and 1 if it was OK.

ic_geo_midsurface *args*

Creates midsurfaces. Arguments are:

- **max_dist***val*: the maximum distance between surface pairs
- **family***val*: the family name for midsurfaces
- **surfaces***names*: the names of the surfaces to compress
- **surfaces2***names*: the names of the second set of surfaces to compress
- **save_old***val* : save compressed surfaces if set
- **partial***val*: create partial midsurface if set
- **similar***val*: do similar pairs only if set
- **alternate***val*: do alternate order of surfaces if set
- **prefer_connected***val*: prefer connected surface pairs if set
- **precision***val*: precision value for midsurface if set

- **offset***val*: just offset side 1 half distance to side 2
- **tolerance***val* : the midsurface tolerance if set
- **ask***val* : quiet operation, if not set, ask yes/no if 1, present the candidates and ask yes/no if 2, just count and return number of candidates if 3

Return value is 0 if there was an error and 1 if it was OK.

ic_geo_lookup *types how spec* [""]

Looks up geometry objects based on certain criteria. *type* may be one of the geometry type names or *all*. This always returns a list of type/name pairs. Useful values for *how* and *args* are:

- **names***names*: return objects with the given names
- **families***names*: return objects with the given families
- **all**: return all objects of the specified types
- **patterns***pats*: looks up objects based on glob-style name matching (for example, FAM*)
- **visible**: all visible objects
- **blanked**: all non-visible objects

ic_geo_get_entity_types *entnames*

For the given list of entities, return a *flat* list of "type entname" pairs, i.e. {type_1 ent_1 type_2 ent_2 ... type_n ent_n}

ic_geo_memory_used

Specifies how much memory is used for the geometry data.

ic_geo_project_mode *which*

Sets the projection mode.

ic_csystem_get_current

Specifies the current coordinate system.

ic_csystem_set_current *what*

Sets the current coordinate system.

ic_csystem_list

Lists the existing coordinate systems.

ic_csystem_get *name*

Returns information on the named coordinate system. This returns a list of 4 items: the type of the coordinate system, the origin, and the 3 vectors that define coordinate system. The type can be one of:

- **cartesian**
- **cylindrical**
- **spherical**
- **toroidal**

ic_csystem_delete *name*

Deletes the named coordinate system.

ic_csystem_create *name type center axis0 axis1 axis2*

Creates a new coordinate system with the given parameters.

- *name*: the name of the system to create.
- *type*: the type which can be one of *cartesian*, *cylindrical*, *spherical*, or *toroidal* (as yet unsupported).
- *center*: the center point in 3-D coordinates.
- *axis0*: the first axis vector.
- *axis1*: the second axis vector.
- *axis2*: the third axis vector.

ic_coords_into_global *pt system* [""]

Translates coordinates from the current or given system into the global system.

ic_coords_dir_into_global *pt system* [""]

Translates a vector from the current or given system into the global system.

ic_coords_into_local *pt system* [""]

Translates coordinates from the global system into the current or given local system.

ic_csystem_display *name on*

Displays the specified coordinate system. If *name* is *all* and *on* is 0 then erase all coordinate systems.

ic_geo_untrim_surface *surf*

Untrims a surface.

ic_geo_get_thincuts

Returns the thincut data.

ic_geo_set_thincuts *data*

Sets the thincut data.

ic_geo_get_periodic_data

Returns the periodic data.

ic_geo_set_periodic_data *data*

Sets the periodic data.

ic_geo_get_family_param *fam name*

Returns the family parameters.

ic_geo_set_family_params *fam args*

Sets the family parameters. If there is no such family, nothing will be done.

ic_geo_reset_family_params *fams params*

Reset family parameters on families *fams* for parameters *params*.

ic_geo_delete_unattached *fams* [""] *quiet* [0] *only_if_dormant* [0]

Deletes unattached geometry.

ic_geo_remove_feature *curves*

Removes features.

ic_geo_merge_curves *curves*

Merges curves.

ic_geo_modify_curve_reappr *curves tol ask* [1] *quiet* [0]

Reapproximates curves.

curves	list of curve names
tol	re-approximation tolerance
ask	prompt to accept result
quiet	suppresses messages
return	list of new curve names

Note:

In interactive mode, if the *ask* parameter is 1 (default), the application prompts you to confirm whether the result is okay for each curve. Parameter is ignored in batch mode.

ic_geo_modify_surface_reappr *surfaces tol ask* [1] *each* [0] *curves* [""]

Reapproximates surfaces.

surfaces	list of surface names
tol	re-approximation tolerance
ask	prompt to accept result if set
each	re-approximate each surface separately if set
curves	list of curve names
return	list of new surface names

Note:

In interactive mode, if the *ask* parameter is 1 (default), the application prompts you to confirm whether the result is okay for each surface. Parameter is ignored in batch mode.

ic_geo_reset_data_structures

Resets the **Tcl** data structures after making big changes to the proj database.

ic_geo_params_update_show_size *type size*

Modifies the display of the size icons for ref, natural, and max size. Can also be used for per-object parameters like tetra_size.

ic_geo_stlrepair_holes *type segs add_nodes int_surf complete_edges* [1] *dcurves* [""] *toler* [""] *part* [""]

Repairs holes using the **stlrepair** functionality. *type* indicates the entity type by which segments are specified, e.g. surface.

ic_geo_stlrepair_edges *type segs merge_tol merge_ends* [-1]

Repairs edges using the **stlrepair** functionality. *type* indicates the entity type by which segments are specified, e.g. surface.

ic_show_geo_selected *type names on color* [""] *force_visible* [0]

Displays some geometry selected or not.

ic_reset_geo_selected

Resets all selection display.

ic_get_geo_selected

Returns all current geometry selections.

ic_set_geo_selected *selected on*

Sets all previous geometry selections.

ic_select_geometry_option

Returns the previously used selection option.

ic_geo_add_segment *type name item pts*

Adds segments or triangles to a surface or curve.

ic_geo_delete_segments *type name item pts*

Deletes segments or triangles from a curve or surface.

ic_geo_restrict_segments *type name item pts*

Restrict to segments or triangles from a surface or curve.

ic_geo_split_segments *type name item how pts*

Splits some segments in a surface or curve.

ic_geo_split_edges *type name pts*

Splits some edges in a surface.

ic_geo_split_one_edge *type name ed*

Splits one edge in a surface.

ic_geo_swap_edges *type name pts*

Swaps some edges in a surface.

ic_geo_move_segments *type name1 name2 item pts*

Moves some segments from one surface to another.

ic_geo_move_node *type name nodes args*

Moves a node in a surface or curve. *nodes* is a list of the node numbers. After this, specify either one or more positions. If one, then all nodes are moved to that position. If more, then the nodes are moved to their corresponding positions.

ic_geo_merge_nodes *type name nodes*

Merges nodes in a surface or curve.

ic_geo_merge_nodes_tol *type name tol*

Merges nodes in a surface or curve by tolerance.

ic_geo_merge_surfaces *to from*

Merges multiple surfaces.

ic_geo_merge_objects *type dest objs*

Merges multiple curves, or surfaces.

ic_geo_merge_points_tol *pts tol*

Merges multiple points using a tolerance.

ic_geo_finish_edit *type name*

Cleans up a surface or curve after editing operations.

ic_geo_delete_if_empty *type name*

Deletes a surface or curve if it is empty.

ic_geo_smallest_segment *type name*

Returns the smallest triangle in a surface.

ic_geo_get_config_param *type name param*

This is kind of an escape.

ic_geo_set_config_param *type name param val*

This is kind of an escape.

ic_geo_set_tag *type names tagname on*

Sets the given tag on the objects, or removes it. If the *tagname* is *pickable* this affects the geometry selection code. If the *type* is clear then the tag is removed from all objects and the *name* and *on* parameters are ignored. If *name* is an empty string then all the objects of that type will be modified.

ic_geo_highlight_segments *type name add hsmode segs*

Highlights some segments of an image.

ic_geo_bounding_box *objlist*

Returns the bounding box of a set of objects. *objlist* is a list of type names pairs, e.g. *{ { curve {C1 C2 C2} point {C2 C3}} }* It can also be **all** which will give the bounding box of all the geometry.

ic_geo_bounding_box2 *objlist*

This is the more rigorous version of the boundary box calculation.

ic_geo_model_bounding_box

This gives the bounding box of all objects in projlib.

ic_geo_feature_size *type name*

Returns the feature size of an object.

ic_geo_replace_surface_mesh *name pts tris*

Replaces a surface mesh. *pts* is a list of x y z triples. *tris* is a list of 3 point numbers. e.g., `ic_geo_replace_surface_mesh SURF.1 {{0 0 0} {1 1 1} ...} {{0 1 2} ...}`

ic_geo_replace_curve_mesh *name pts bars*

Replaces a curve mesh. *pts* is a list of x y z triples. *bars* is a list of 2 point numbers. e.g., **`ic_geo_replace_curve_mesh CRV.1 {{0 0 0} {1 1 1} ...} {{0 1} {2 3} ...}`**

ic_geo_vec_diff *p1 p2*

Computes the displacement vector between two points.

p1	point -- e.g. {1 2 3}
p2	point -- e.g. {3 4 3}
return	3-tuple containing difference

Example

```
   # Compute the dot product between two vectors
 #
 if [catch {ic_geo_vec_diff {1 2 3} {3 4 3} } crv1] {
 mess "$crv1\n" red
 } else {
 mess "vec diff, $crv1\n"
 }
```

ic_geo_vec_dot *v1 v2*

Computes the dot product between two vectors.

v1	vector -- e.g. {1 2 3}
v2	vector -- e.g. {3 4 3}
return	dot product

Example

```
   # Compute the dot product between two vectors
 #
 if [catch {ic_geo_vec_dot {1 2 3} {3 4 3} } crv1] {
 mess "$crv1\n" red
 } else {
 mess "vec dot, $crv1\n"
 }
```

ic_geo_vec_mult *v1 v2*

Computes the cross product between two vectors.

v1	vector -- e.g. {1 2 3}
v2	vector -- e.g. {3 4 3}
return	cross product

Example

```
   # Compute the cross product between two vectors
 #
 if [catch {ic_geo_vec_mult {2 0 1} {1 3 0} } crv1] {
 mess "$crv1\n" red
 } else {
 mess "vec mult, $crv1\n"
 }
```

ic_geo_vec_nrm *vec*

Normalizes a vector.

vec	vector -- e.g. {1 2 3}
return	normalized vector

Example

```
   # Normalize a vector
 #
 if [catch {ic_geo_vec_nrm {2 2 4} } crv1] {
 mess "$crv1\n" red
 } else {
 mess "vec normalize, $crv1\n"
 }
```

ic_geo_vec_len *vec*

Computes the length of a vector.

vec	vector -- e.g. {1 2 3}
return	length

Example

```
   if [catch {ic_geo_vec_len {3 4 0} } len] {
 mess "$len\n" red
 } else {
 mess "vec length should be 5: $len\n"
 }
```

ic_geo_pnt_dist *pnt1 pnt2*

Computes the distance between two points.

pnt1	point -- e.g. {1 2 3} or point name
pnt2	point -- e.g. {1 2 3} or point name
return	distance between points

Example

```
   #
 if [catch {ic_geo_pnt_dist {2 2 4} {3 2 1}} len] {
 mess "Error: $len\n" red
 } else {
 mess "Distance = $len\n"
 }
```

ic_geo_vec_smult *vec scal*

Multiplies a vector by a scalar.

vec	vector -- e.g. {1 2 3}
scal	scalar -- e.g. 42
return	scalar product vector

Example

```
   # Multiply a vector by a scalar
 #
 if [catch {ic_geo_vec_smult {1 2 3} 42 } crv1] {
 mess "$crv1\n" red
 } else {
 mess "vec smult, $crv1\n"
 }
```

ic_geo_vec_sum *v1 v2*

Computes the sum of two vectors.

v1	vector -- e.g. {1 2 3}
v2	vector -- e.g. {3 4 3}
return	cross product

Example

```
   # Compute the sum of two vectors
 #
 if [catch {ic_geo_vec_sum {1 2 3} {4 5 6}} crv1] {
 mess "$crv1\n" red
 } else {
 mess "vec sum, $crv1\n"
 }
```

ic_geo_crv_length *crvs t_min* [0] *t_max* [1]

Computes the arc length of a curve segment.

crvs	list of one or more curves
t_min	lower limit of segment (unitized
t_max	upper limit of segment (unitized
return	list of computed arc lengths

Notes:

- *t_min* and *t _max* default to 0 and 1 respectively
- For example usage, refer to `ic_geo_crv_length.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_srf_rev *family name gen base zaxis srtang endang dxn* [a] *bld_topo* [0]

Creates a revolution surface from a generator curve and axis.

family	family containing surface
name	name of created surface

gen	generator curve(s)
base	axis base point
zaxis	axis direction vector
srtang	start angle (degrees)
endang	end angle (degrees)
dxn	c-clockwise, a-anticlockwise
bld_topo	associate edge curves
return	name of created surface

Notes:

- The specified surface name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- Positions may be specified explicitly or using names of prescribed points
- The *dxn* flag determines whether the curve is swept clockwise or anti-clockwise (counter clockwise) about the rev axis.
- For example usage, refer to `ic_geo_cre_srf_rev.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_crv_iso_crv *family name srfs par sel do_split* [0] *coord* [0]

Creates isoparametric curve segments on a surface.

family	family containing curves
name	name of created curves
srfs	list of surface names list of surface names
par	surface parameter 0 <= par <= 1
sel	== 0 u cons; == 1 v cons
do_split	== 1 split the surface
coord	== 0 use restricted coordinates; == 1 use natural coordinates
return	list of created curves/surfaces

Notes:

- The defining parameter is assumed to be unitized
- When applied to trimmed surfaces
  - The feature may produce multiple result curves
  - Restricted coordinates are taken with respect to the active region of the trimmed surface, not the domain of the underlying surface.

  - Natural coordinates are taken with respect to the underlying surface. This alternative is consistent with the output of **ic_geo_find_nearest_srf_pnt** in that while the coordinates are still unitized, they run through the full extent of the underlying surface.

- The specified curve name may be modified to resolve name collisions
- If the function returns with the error status set, the result string will contain an error message.
- The surface parameter is unitized.
- The return value is a list containing two elements, names of created curves and names of created surfaces
- For example usage, refer to `ic_geo_cre_crv_iso_crv.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_srf_pln_3pnts *family name p1 p2 p3 rad*

Creates a bspline plane from 3 points.

family	family containing plane
name	name of created plane
p1	point data, e.g. {1 2 3}
p2	point data, e.g. {1 2 3}
p3	point data, e.g. {1 2 3}
scale	scales surface extents

Notes:

- The specified surface name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- For an annotated example of usage, refer to `ic_geo_cre_srf_pln_3pnts.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_srf_pln_nrm_pnt *family name pnt nrm rad*

Creates a bspline plane from a point and normal vector.

family	family containing plane
name	name of created plane
pnt	point data, e.g. {1 2 3}
nrm	plane normal, e.g. {1 1 1}
rad	radius of created surface

Notes:

- The specified surface name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- For an annotated example of usage, refer to `ic_geo_cre_srf_pln_nrm_pnt.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_srf_pln_nrm_dist *family name nrm dist rad*

Creates a bspline plane from normal vector at a distance from origin.

family	family containing plane
name	name of created plane
nrm	plane normal, e.g. {1 1 1}
dist	signed distance between origin and plane
rad	radius of created surface

Notes:

- The specified surface name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- This variant replaces plane definition by coefficients from older API's. The plane equation can be written in terms of the input variables as:

```
DOT(nrm, X) = LENGTH(nrm)*dist
```

Example

```
   if [catch {ic_geo_cre_srf_pln_nrm_dist duck dewey \
{1 2 3} {1 0 0} 10 42} pln1] {
mess "$pln1\n" red
} else {
mess "created a plane, $pln1\n"
}
```

ic_geo_cre_arc_from_pnts *family name p1 p2 p3*

Create a bspline arc from 3 points.

family	family containing curve
name	name of created curve

p1	point data, e.g. {1 2 3}
p2	point data, e.g. {1 2 3}
p3	point data, e.g. {1 2 3}

Notes:

- The specified curve name may be modified to resolve name collisions.
- Positions may be specified explicitly or using names of prescribed points.
- If the function returns with the error status set, the result string will contain an error message.
- For an annotated example of usage, refer to `ic_geo_cre_arc_from_pnts.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_bsp_crv_n_pnts *family name pnts tol* [0.0001] *deg* [3]

Creates a bspline curve from n points.

family	family containing curve
name	name of created curve
pnts	point data
tol	approximation tolerance
deg	curve degree = 1 (linear) or 3 (cubic)

Notes:

- The specified curve name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- Positions may be specified explicitly or using names of prescribed points.
- The approximation tolerance is relative. It will be scaled by the pointset chordlength to form an absolute tolerance. It has a default value of 0.0001.
- For example usage, refer to `ic_geo_cre_bsp_crv_n_pnts.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_bsp_crv_n_pnts_cons *family name pnts fixPnts tanCons tanIndx tol* [0.001]

Creates a bspline curve from n points *with constraints*.

family	family containing curve
name	name of created curve
pnts	point data
fixPnts	fixed points
tanCons	specified tangents
tanIndx	indices of points in pnts associated to tangent vectors

tol	approximation tolerance

Notes:

- The specified curve name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- Positions may be specified explicitly or using names of prescribed points
- Point array is indexed as 0, 1, 2, . . .
- The approximation tolerance is relative. It will be scaled by the pointset chordlength to form an absolute tolerance. It has a default value of 0.0001.
- For example usage, refer to `ic_geo_cre_bsp_crv_n_pnts_cons.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_crv_arc_ctr_rad *family name center x_ax normal radius srtang endang*

Creates a bspline arc from center, radius information.

family	family containing curve
name	name of created curve
center	arc center
x_ax	vector aligned along angle == 0
normal	arc normal
radius	arc radius
srtang	start angle (degrees)
endang	end angle (degrees)
return	name of created curve

Notes:

- The specified curve name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- If endang < srtang or (endang - srtang) > 360, the angle will be adjusted by adding 360 increments.
- Positions may be specified explicitly or using names of prescribed points.
- For annotated examples of usage, refer to `ic_geo_cre_crv_arc_ctr_rad.tcl` and `ic_geo_create_surface_from_curves.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_srf_cyl *family name center x_ax z_ax radius srtang endang length*

Create a bspline cylinder from center, radius information.

family	family containing surface
name	name of created surface
base	cylinder base point
x_ax	vector aligned along angle == 0
z_ax	vector aligned along cyl axis
radius	cylinder radius
srtang	start angle (degrees)
endang	end angle (degrees)
length	length
return	name of created surface

Notes:

- The specified surface name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- If endang < srtang or (endang - srtang) > 360, the angle will be adjusted by adding 360 degree increments.
- Positions may be specified explicitly or using names of prescribed points.
- For an annotated example of usage, refer to `ic_geo_cre_srf_cyl.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_line *family name p1 p2*

Create a bspline line from 2 points.

family	family containing curve
name	name of created curve
p1	point data, e.g. {1 2 3}
p2	point data, e.g. {1 2 3}
return	name of created curve

Notes:

- The specified curve name may be modified to resolve name collisions
- If the function returns with the error status set, the result string will contain an error message.
- Positions may be specified explicitly or using names of prescribed points
- For an annotated example of usage, refer to `ic_geo_cre_line.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_pnt *family name pnt in_lcs* [1]

Creates a prescribed point from coordinates.

family	family containing point
name	name of created point
pnt	point data, e.g. {1 2 3}
in_lcs	1 if the location should be in the current local coordinate system (default)
return	name of created point

Notes:

- The specified point name may be modified to resolve name collisions
- If the function returns with the error status set, the result string will contain an error message.

Example

```
   # Create a prescribed point from coordinates

 if [catch {ic_geo_cre_pnt duck louie {1 2 3} } pnt1] {
 mess "$pnt1\n" red
 } else {
 mess "created a ducky point, $pnt1\n"
 }
```

ic_geo_cre_mat *fam name pt in_lcs* [1]

Create a material point from coordinates.

family	family containing material point
name	name of created material point
pnt	point data, e.g. {1 2 3}, or the word outside
in_lcs	1 if the location should be in the current local coordinate system (default)
return	name of created material point

Notes:

- The specified point name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.

Example

```
   # Create a material point from coordinates
```

```
if [catch {ic_geo_cre_mat duck louie {1 2 3} } pnt1] {
mess "$pnt1\n" red
} else {
mess "created a ducky point, $pnt1\n"
}
```

ic_geo_get_srf_nrm *upar vpar srf*

Get the normal vector of a surface at a parameter.

upar	surface u parameter
vpar	surface v parameter
srf	list of surfaces to evaluate
return	list of 3-tuple of doubles

Notes:

- If the function returns with the error status set, the result string will contain an error message.
- The return string will be a list of 3-tuples
- The input uv coordinates should be unitized.
- For example usage, refer to `ic_geo_get_srf_nrm.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_get_srf_pos *upar vpar srf*

Get a surface position at a parameter.

upar	surface u parameter
vpar	surface v parameter
srf	list of surfaces to evaluate
return	list of 3-tuple of doubles

Notes:

- If the function returns with the error status set, the result string will contain an error message.
- The return string will be a list of 3-tuples
- The input uv coordinates should be unitized.
- For example usage, refer to `ic_geo_get_srf_pos.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_pnt_on_srf_at_par *family name upar vpar srf*

Creates a prescribed point on a surface at a parameter.

family	family containing point
name	name of created point
upar	surface u parameter
vpar	surface v parameter
srf	list of surfaces to evaluate
return	names of created points

Notes:

- The specified point name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- The input uv coordinates should be unitized.
- For example usage, refer to `ic_geo_cre_pnt_on_srf_at_par.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_pnt_on_crv_at_par *family name par crv*

Creates a prescribed point on a curve at a parameter.

family	family containing point
name	name of created point
par	curve parameter
crv	name of curve to evaluate
return	name of created point

Notes:

- The specified point name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- For example usage, refer to `ic_geo_cre_pnt_on_crv_at_par.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_crv_concat *family name tol crvs*

Create a new curve by concatenating existing curves.

family	family containing curve
name	name of created curve
tol	merge tolerance

crvs	list of curves to be joined
return	name of created curve

Notes:

- The specified curve name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- The list *crvs* contains curve *curve names*.
- For example usage, refer to `ic_geo_cre_crv_concat.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_create_curve_concat *family name tol crvs*

Deprecated version of **ic_geo_cre_crv_concat**. This function has been replaced by **ic_geo_cre_crv_concat** (p. 76).

ic_geo_cre_srf_from_contour *family name tol crvs*

Create a new surface spanning a planar contour.

family	family containing new surface
name	base name of created surface
tol	merge tolerance
crvs	list of curves to span

Notes:

- The specified surface name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- The list *crvs* contains curve *curve names*.
- For example usage, refer to `ic_geo_cre_srf_from_contour.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_create_surface_from_curves *family name tol crvs bld_topo* [0]

Create a new surface spanning two to four curves.

family	family containing new surface
name	name of created surface
tol	merge tolerance
crvs	list of curves to span

bld_topo	associate edge curves

Notes:

- The specified surface name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- The list *crvs* contains curve *curve names*.
- For example usage, refer to `ic_geo_create_surface_from_curves.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_create_param_surface *family name nu nv ord_u ord_v rational u_knots v_knots control_pts loops*

Create a new surface from a u,v set of coordinates.

family	family containing new surface
name	name of created surface
nu	number of u coordinates
nv	number of v coordinates
ord_u	order of u
ord_v	order of v
rational	dimension of polynomial fit above 3
u_knots	for rational=0, list of u coordinates
v_knots	for rational=0, list of v coordinates
control_pts	points the surface should go through
loops	loops to trim

- If the function returns with the error status set, the result string will contain an error message.

ic_geo_list_crv_data *file format crvs*

Lists the IGES data defining a list of curves.

out_file	output file
format	output format
crvs	list of curves

Notes:

- Output formats:

iges	IGES Style
tetin	TETIN Style

- If the function returns with the error status set, the result string will contain an error message.
- The list *crvs* contains *curve names*.

Example

```
   # List the IGES data defining a list of curves
#
if [catch {ic_geo_cre_crv_arc_ctr_rad duck dewey \
{1 2 3} {1 0 0} {1 1 1} 4.2 0 1.047197} crv1] {
mess "$crv1\n" red
} else {
mess "created a ducky arc, $crv1\n"
if [catch {ic_geo_cre_crv_arc_ctr_rad duck louie \
{1 2 3} {0 1 0} {1 1 1} 4.2 0 1.047197} crv2] {
mess "$crv2\n" red
} else {
mess "created a ducky arc, $crv2\n"
if [catch {ic_geo_list_crv_data out.txt iges \
"$crv1 $crv2"} err] {
mess "$err\n" red
}
}
```

ic_geo_list_srf_data *file format srfs*

Lists the IGES data defining a list of surfaces

out_file	output file
format	output format
crvs	list of surfaces

Notes:

- Output formats:

iges	IGES Style
tetin	TETIN Style

- If the function returns with the error status set, the result string will contain an error message.
- The list *srfs* contains *surface names*.

ic_geo_make_conn_regions *type entities surf_angle* [180] *surf_curvature* [360]

Makes connected regions of type: surface or curve entities are a list of the type, surfaces or curves. If *type* is surface, *surf_angle* limits connectivity based on curves over the feature angle; default is 180, 0 would make each surface separate. If type is surface, *surf_curvature* limits connectivity to surfaces with curvature over value; default is 360, 0 would make each surface separate. The return is a separated list based on connectivity.

ic_geo_get_attached_entities *attach_type orig_type entities*

Gets all attached entities based on *attach_type* and *orig_type* to a list of entities. *attach_type* could be boundary or incident. *orig_type* could be surface curve or point. Example: if you want all curves attached to a list of surfaces *attach_type* is boundary, *orig_type* is surface, *entities* is the list of surfaces.

ic_geo_get_entities_by_attach_num *type num entities* [""]

Gets all entities of a type: point or curve; that have defined number of entities attached to it. For example a single curve has 1 entity attached. *entities* is list of type to look for. Default is all entities of this type. If *num* is multiple, it will find attachments of 3 or more. If *num* is double, it will find attachments of 2. If *num* is single, it will find attachments of 1. If *num* is unattached, it will find attachments of 0.

ic_geo_get_internal_surface_boundary *surf not_single* [0]

This command will take a given "surf" and return the curves that are internal. In other words, it will return all attached curves except those on outer boundary. Optional argument *not_single* will limit the returned curves to only those that are attached to more than 1 surface.

ic_geo_find_internal_outer_loops *surfs not_single* [0] *all_boundary* [0]

This procedure returns a list of outer curves and inner curves attached to a set of surfaces, optional argument *not_single* will limit the list to just curves attached to more than 1 surface.

ic_geo_find_internal_surfaces *loop surrounding_surfs outer_curves* [""] *exclusion_surfs* [""]

This function will find a set of surfaces enclosed by a loop of curves.

ic_geo_make_conn_buttons *loop exclusion_surfs* [""]

This function will take a curve list (loop), and find all surfaces attached to it excluding any given *exclusion_surfs*.

ic_geo_split_surfaces_at_thin_regions *srfs tolerance min_res_curve_len*

Splits boundaries of the given surfaces at thin regions, that is, where a surface boundary points is less than **tolerance** from an other boundary curve. It will not, however, split curves which would result in segments of length less than *min_res_curve_len*. Returns a list of all new points, if any.

ic_geo_surface_create_smart_nodes *srfs tolerance min_res_curve_len*

Split boundaries of the given surfaces at thin regions, that is, where a surface boundary points is less than *tolerance* from an other boundary curve. It will not, however, split curves which would result in segments of length less than *min_res_curve_len*. Returns tcl-error-stat.

ic_geo_surface_topological_corners *surfs*

For each surface in the given list *surfs* returns a list of the four corners of a rectangular topology of that surface. The points are ordered around the rectangular either clockwise or counter-clockwise. The form of the list returned is: **`"{surf_name_1 {pt_name_1_1 pt_name_1_2 pt_name_1_3 pt_name_1_4} {surf_name_2 {pt_name_2_1 ...}}"`**

ic_geo_flanges_notch_critical_points *surfs*

Returns the critical point of the notch in a given flange surface.

ic_geo_trm_srf_at_par *srf par sel*

Splits a surface at a parameter.

srf	surface name
par	surface parameter 0 <= par <= 1
sel	== 0 u cons; == 1 v cons

Example

```
   if [catch {ic_geo_trm_srf_at_par $srf1 0.5 0} err] {
 mess "$err\n" red
 }
```

ic_geo_trm_srfs_by_curvature *srfs ang*

Splits folded surfaces by maximum curvature.

srfs	surface names
ang	maximum total curvature

ic_surface_curvature *surf tol* [100] *debug* [0]

Calculates curvature of surface.

surf	surface name
btol	relative boundary tolerance (100 -> 1/100 -> 1%)

ic_hull_2d *entities tol* [0] *four* [1] *type* [surface] *shrink* [0] *debug* [0]

Creates 2D hull of surfaces or curves.

entities	entity names
tol	approximation tolerance
four	split hull at best four corners if set
type	surface or curve
shrink	relative shrink tolerance (0 ... 1)

ic_surface_from_points *points part* [""] *name* [""]

Creates a faceted surface from points using a 2D Delaunay triangulation.

points	point names

ic_geo_surface_extend *curve surfaces toler* [0] *bld_topo* [1] *perpendicular* [1] *connect* [0] *concat_crvs* [1] *db* [0]

Extends surface edge to surface(s).

curve	"yellow" edge to extend
surfaces	surfaces to extend surface edge to
toler	geometry tolerance
bld_topo	associate edge curves
perpendicular	extend normal to curve if 1, create a curtain surface if 0, or do a parametric extension if 2
connect	connect extension to target surface(s) if set
concat_crvs	clean points on surface edges if set

ic_geo_cre_srf_crv_drv_srf *family name gencrv ctrcrv bld_topo* [0]

Create a curve driven surface.

family	family containing new surface
name	name of created surface
gencrv	name of generator or driven curve
ctrcrv	name of center or driver curve
bld_topo	associate edge curves

Notes:

- The specified surface name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.

ic_geo_get_types *which* [all]

This function returns a list of all geometric entity types available in the loaded geometry. If no geometry is loaded, it returns all possible types: **"surface curve point material density loop"**

ic_flood_fill_surface_angle *surf curve angle*

Returns the list of incident surfaces at curve whose dihedral angle with surf is less than the feat_angle.

ic_geo_flood_fill *what ents all* [1] *feat_angle* [0] *bound_mode* [all] *nedges* [0]

Returns the list of entities connected by the lower dimension entities. For example surfaces connected by the curves.

- *what* - 'curve' or 'surface'
- *ents* - list of type, name pairs
- *all* - 0 if only one level is desired
- *feat_angle* - 0 >= theta <= 90, valid only for $what = surface
- bound_mode - 'outer' if only outer loop is desired, valid only for $what = surface
- *nedges* - for use with curves - if 0 then all attached curves, else only go with curves that have the specified number of attached surfaces

ic_geo_get_triangulation_tolerance

Returns a two-element list containing the triangulation tolerance of the model, such that the first element is the tolerance (real number) and the second element is an integer (0 or 1) indicating whether or not the value is relative to a global setting.

ic_geo_convex_hull *entities name fam*

Creates the convex hull of the objects. *entities* is a list of pairs, where the first element is the type and the second is the name.

ic_geo_remove_triangles_on_plane *surf plane tol*

Remove triangles on a plane in the named surface.

ic_geo_bbox_of_entities *ents*

Return the bounding box of some objects. The *ents* argument is a list of the form **`{{type name} {type name} ...}`**

ic_geo_classify_by_regions *planes entities how*

Used by convex hull.

ic_geo_split_surfaces *surfs planes*

Used by convex hull.

ic_geo_elem_assoc *domain assoc*

Generate mesh geometry associativity for CATIA interface.

domain	domain file

assoc	output associativity file

ic_geo_cre_bsp_srf_by_pnt_array *family name n_ptu n_ptv pnts tol* [0.0001]

Creates a bspline surface from a point array.

family	family containing surface
name	name of created surface
n_ptu	number of points in u direction
n_ptv	number of points in v direction
pnts	point data

Notes:

- The specified surface name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- Positions may be specified explicitly or using names of prescribed points
- The approximation tolerance is relative. It will be scaled by the pointset radius to form an absolute tolerance. It has a default value of 0.001.

ic_geo_cre_geom_input *in_file fit_tol* [0.0001] *mode* [input] *pnt_fam* [PNT] *pnt_prefix* [pnt] *crv_fam* [CRV] *crv_prefix* [crv] *srf_fam* [SRF] *srf_prefix* [srf]

Create point and b-spline geometry from an Ansys ICEM CFD Input file.

fit_tol	approximation tolerance
mode	type of ICEM CFD input file
pnt_fam	family for point entities
pnt_prefix	prefix for point names
crv_fam	family for curve entities
crv_prefix	prefix for curve names
srf_fam	family for surface entities
srf_prefix	prefix for surface names

Notes:

- If the function returns with the error status set, the result string will contain an error message.
- MED must be initialized by loading a tetin file or opening an empty part (see **ic_empty_tetin** (p. 35)).

- The approximation tolerance is relative. It will be scaled by the pointset radius to form an absolute tolerance. It has a default value of 0.001.
- Supported values for *mode* are **input** and **plot3d**. **named** for named entities is planned, but not yet supported.
- For example usage, refer to `ic_geo_cre_geom_input.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_import_str_to_cad *doms srf_fam* [SRFS] *crv_fam* [""] *pnt_fam* [""]

Converts structured surface domains to b-spline geometry. If successful, the current mesh and geometry are unloaded, and the new geometry is loaded. Surfaces, curves, and points are created only if families are provided for each argument type. By default, only surfaces will be created.

ic_geo_crv_data *crvs datums*

Return the b-spline data associated to a curve.

crvs	list of curve to examine
datums	list of curve properties to return

Notes:

- curve properties may be a list of one or more of the following:
  - *order* — Integer order of spline
  - *ncp* — Number of control points (3-tuples in model space)
  - *rat* — Rational flag -- 1 if rational; 0 if integral
  - *cps* — Return model space control points
  - *knots* — Return knot vector
  - *weights* — Return a list of curve weights
- If the utility is called for multiple curves, the data for each curve will be grouped together
- If weights are requested for an integral spline, the list of weights returned will be null
- For example usage, refer to `ic_geo_crv_data.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_srf_data *srfs datums*

Return the b-spline data associated to a surface

srfs	list of surface to examine
datums	list of surface properties to return

Notes:

- surface properties may be a list of one or more of the following:
  - *order* — Integer orders (u and v) of spline
  - *ncp* — Control point counts in u and v
  - *rat* — Rational flag -- 1 if rational; 0 if integral
  - *cps* — Return model space control points. Control points are returned in a list arranged in v-major order (see example program for details)
  - *knots_u* — Return u knot vector
  - *knots_v* — Return v knot vector
  - *weights* — Return a list of surface weights. Weights are returned in a list arranged in v-major order
- If the utility is called for multiple surfaces, the data for each surface will be grouped together
- If weights are requested for an integral spline, the list of weights returned will be null
- For example usage, refer to `ic_geo_srf_data.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_srf_loft_crvs *family name tol crvs sec_ord* [4] *form* [0] *bld_topo* [0]

Create a new surface by lofting two or more curves

family	family containing new surface
name	name of created surface
tol	approximation tolerance
crvs	list of curves to loft
sec_ord	order in cross direction
form	0 (C1 cubic blend) or 1 (C2 cubic blend)
bld_topo	associate edge curves

Notes:

- The surface order in the cross direction must be 2 (linear) or 4 (cubic).
- The specified surface name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- The approximated surface should lie within *tol* of the curves.
- The list *crvs* contains curve *curve names*.

ic_geo_cre_crv_test_project_surface *family name surface curve dir*

Project a curve to a surface.

family	family for new curve
name	name for new curve (can be "")
surface	name of input surface
curve	name of input curve
dir	direction vector

Return value is the name of the new curve.

ic_geo_cre_surface_section *family name surface mode P0 P1 P2* [""] *trim* [0]

Create curve as section of a surface with plane, cylinder or segment.

family	family for new curve
name	name for new curve (can be "")
surface	name of input surface
mode	0 - section with plane
	1 - section with cylinder
	2 - section with segment
P0, P1, P2	define plane, cylinder or segment:
	Surface through three given points
	Cylinder with axis on P0-P1 line and P2 on the radius
	Segment from P0 to P1 projected in direction P0-P2

Return value is the name of the new curve.

ic_geo_offset *family name surface_to_offset offset max_factor* [3]

Offset surface using mesh representation.

ic_geo_cre_crv_datred *family name crvs tol* [0.001]

Creates a reduced b-spline curve

family	family containing curve
name	name of created curve
crvs	list of curves to be joined
tol	approximation tolerance
return	names of created curves

Notes:

- The specified curve name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- The list *crvs* contains curve *curve names*.
- For example usage, refer to `ic_geo_cre_crv_datred.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_srf_datred *family name srfs tol* [0.001]

Create a reduced b-spline surface.

family	family containing surface
name	name of created surface
srfs	list of surfaces to be joined
tol	approximation tolerance
return	names of created surfaces

Notes:

- The specified surface name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- The list *srfs* contains surface *surface names*.
- For example usage, refer to `ic_geo_cre_srf_datred.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_srf_sweep *family name gen drv bld_topo* [0]

Creates a swept surface from a generator curve and axis.

family	family containing surface
name	name of created surface
gen	generator curve(s)
drv	drive curve or vector
bld_topo	associate edge curves
return	name of created surface

Notes:

- The specified surface name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.

- Positions may be specified explicitly or using names of prescribed points.
- For example usage, refer to `ic_geo_cre_srf_sweep.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_crv_is_opposite *crv1 crv2*

Determines whether two curves are oriented in parallel or opposite directions.

crv1	curve 1 name
crv2	curve 2 name
return	1 if opposite, 0 otherwise

Notes:

- Main use of this function is to diagnose failures in some of the surface construction codes.
- For example usage, refer to `ic_geo_crv_is_opposite.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_crv_is_edge *crv*

Determines whether a curve bounds a surface.

crv	curve name
return	number of surfaces bounded

ic_geo_fix_degen_geom *switch* [0]

Activates repair function for degenerate bsplines in the tetin reader. For now these functions are disabled by default.

switch	0 for off; 1 for on

ic_geo_find_nearest_srf_pnt *srf pnt want_ext* [0]

Finds parameters of closest point on surface.

srf	name of surface
pnt	test point
want_ext	want extended output
return	uv pair

Notes:

- If the function returns with the error status set, the result string will contain an error message.

- The uv coordinates will be unitized.
- For example usage, refer to `ic_geo_find_nearest_srf_pnt.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_find_nearest_crv_pnt *crv pnt*

Finds parameters of closest point on curve.

crv	name of curve
pnt	test point
return	t parameter

Notes:

- If the function returns with the error status set, the result string will contain an error message.
- The t parameter will be unitized.
- For example usage, refer to `ic_geo_find_nearest_crv_pnt.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_distance_from_surfaces *surfs coords*

Gets the distance of *coords* from the nearest surface in **surface_family**

ic_geo_nearest_surface_list *coords surfaces*

Gets nearest surface to *coords* from a list of surfaces.

ic_geo_get_crv_nrm *par crv*

Gets the normal vector of a curve at a parameter.

par	curve t parameter
crv	list of curves to evaluate
return	list of 3-tuple of doubles

Notes:

- If the function returns with the error status set, the result string will contain an error message.
- The return string will be a list of 3-tuples
- The input uv coordinates should be unitized.

ic_geo_get_crv_pos *par crv*

Gets a position on a curve at a parameter.

par	curve t parameter
crv	list of curves to evaluate
return	list of 3-tuple of doubles

Notes:

- If the function returns with the error status set, the result string will contain an error message.
- The input uv coordinates should be unitized.
- The return string will be a list of 3-tuples
- For example usage, refer to `ic_geo_get_crv_pos.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_get_crv_binrm *par crv*

Gets the binormal vector of a curve at a parameter.

par	curve t parameter
crv	list of curves to evaluate
return	list of 3-tuple of doubles

Notes:

- If the function returns with the error status set, the result string will contain an error message.
- The input uv coordinates should be unitized.

ic_geo_cvt_uns_to_bsc *family base uns*

Creates one or more bspline curves from an unstructured curve.

family	family for new curves
base	base name of created curves
uns	name(s) of unstructured curve(s)
return	name of created curves

Notes:

- If the function returns with the error status set, the result string will contain an error message.
- If the input curve is a b-spline curve it will be returned without modification as the output curve
- For example usage, refer to `ic_geo_cvt_uns_to_bsc.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_srf_area *srfs*

Computes the area of one or more surfaces.

srfs	list of one or more surfaces
return	area of surfaces

Notes:

- Surface area is computed from projlib's facetization of the geometry; the results will be influenced by the value of the triangulation tolerance when the part was read.
- For example usage, refer to `ic_geo_srf_area.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_sort_by_srf_area *surf_list args* [""]

Sorts surfaces by their surface area. *args* is arguments for the sort.

ic_geo_reduce_face *srfs*

Trims a surface back to its active area.

srfs	list of one or more surfaces

Notes:

- A form of data reduction; trims the undisplayed portion of a b-spline away.
- Unless the underlying surface is reduced by at least 5% in the u or v coordinate, the surface will be left unmodified.
- For example usage, refer to `ic_geo_reduce_face.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_get_crv_tan *par crv*

Gets the tangent vector of a curve at a parameter.

par	curve t parameter
crv	list of curves to evaluate
return	list of 3-tuple of doubles

Notes:

- The tangent vector will be the un-normalized derivative vector with respect to the unitized parameterization

- If the function returns with the error status set, the result string will contain an error message.
- The input uv coordinates should be unitized.

ic_geo_mod_crv_tanext *crvs dist srtend*

Tangentially extend a curve.

crvs	list of curves to extend
dist	distance (relative) of ext.
srtend	extend start (0), end(1)
return	names of extended curves

Notes:

- The specified curve name may be modified to resolve name collisions
- The length of the extension will be roughly "**`dist*curve length`**"
- If the function returns with the error status set, the result string will contain an error message.
- If the curve has *topology* (i.e. references surfaces or vertices) it will be copied rather than modified.
- The list *crvs* contains curve *curve names*.
- For example usage, refer to `ic_geo_mod_crv_tanext.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_mod_srf_tanext *srfs dist srtend bld_topo* [0]

Tangentially extend a surface.

srfs	list of surfaces to extend
dist	distance (relative) of ext.
edge	index of edge to extend
return	names of modified surfaces

Notes:

- The *dist* parameter is unitless and will be scaled by the lengths of the u or v constant control point rows.
- If the function returns with the error status set, the result string will contain an error message.
- The edge indicator is set as follows:
  - 1 V-Min direction
  - 2 U-Min direction

  – 3 V-Max direction

  – 4 U-Max direction

- The list *srfs* contains surface *surface names*.
- For example usage, refer to `ic_geo_mod_srf_tanext.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_mod_srf_ext *srfs dist edge bld_topo* [0]

Extend a surface.

srfs	list of surfaces to extend
dist	distance of extension
edge	curve at edge to extend
return	names of modified surfaces

ic_geo_mod_crv_match_crv *crv1 crv2 crv1end* [0] *crv2end* [0] *modes* [""]

Matches two curves.

crv1	name of first curve
crv2	name of second curve
crv1end	curve1 end indicator
crv2end	curve2 end indicator
modes	5 element list of flags
return	names of modified curves

Notes:

- The curve end parameters, *crv1end* and *crv2end*, take the following values:

closest endpoint	0
start point	1
end point	2

- The *modes* argument is an optional 5 element list of flags addressing the following functions:

mode[0]	0 point only
	1 point and tangent (default)
	2 point, tangent, and curvature
	3 point and curvature
mode[1]	1 geometric matching (default)

	2 exact matching
mode[2]	0 do not change order
	1 permit change of order (default)
mode[3]	0 both splines matched (default)
	1 only first spline changed
mode[4]	0 3d matching (default)
	1 4d matching

- If the function returns with the error status set, the result string will contain an error message.
- For example usage, refer to `ic_geo_mod_crv_match_crv.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_mod_crv_match_pnt *crv pnt crvend* [0] *modes* [""]

Match curve and a point.

crv	name of curve
pnt	name of point
crvend	curve end indicator
return	name of modified curve

Notes:

- The curve end parameter, *crvend* takes the following values:

closest endpoint	0
start point	1
end point	2

- If the function returns with the error status set, the result string will contain an error message.

ic_geo_cre_srf_offset *family name base offset all_conn* [0] *stitch* [0]

Creates one or more offset surfaces.

family	family containing surface
name	base name of created surface
base	surface(s) to offset
offset	distance to offset
all_conn	offset connected surfs if set

stitch	preserve connected edges if set
return	name(s) of created surface(s)

Notes:

- The specified surface name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- Negative offsets are allowed.
- If more than one surface is given to the routine, offsets will be oriented relative to the first surface if the two surfaces are related by an edge adjacency chain.
- If the *all_conn* flag is set, all surfaces connected to the first surface by an edge adjacency chain will be offset.
- If the *stitch* flag is set, the offsets of two surfaces sharing an edge will be extended tangentially so that the offset surfaces also share an edge
- For example usage, refer to `ic_geo_cre_srf_offset.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_build_bodies *fam* [LIVE] *buildtopo* [0] *tol* [0.01] *multi* [0] *newm* [0] *surf* [""] *assem* [0] *from_solids* [0]

Automatically creates a body for each closed volume of surfaces as determined by the connectivity produced from build topology.

fam	Family for bodies
buildtopo	Build topology if non-zero
tol	Tolerance for the optional build topology function
multi	Old style assembly naming if non-zero
newm	Use the new schema if non-zero
surf	Initial surface

ic_geo_create_volume *matlpt name* [""] *fam* [LIVE]

Creates volume from material point name, *matlpt*.

ic_geo_reset_bodies

Updates the current defined bodies in the model, by removing nonexistent ones and adding any new ones to the display.

ic_geo_create_body *surfs name* [""] *fam* [""] *quiet* [0]

Creates a body from the collection of surfaces, *surfs*. The new body will be given the name, *name*, in the family, *fam*.

ic_geo_get_body_matlpnt *bdy*

Returns the material point name associated with the body, *bdy*.

ic_geo_srf_radius *srfs*

Computes the *radius* of a b-spline surface.

srfs	list of one or more surfaces
return	list of computed surface radii

Notes:

- The radius is the maximum chord length of the control point rows.
- For example usage, refer to `ic_geo_cre_srf_offset.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_srf_offset_edge *family name crv offset*

Creates an offset surface from a generator curve and axis.

family	family containing surface
name	name of created surface
gen	generator curve(s)
base	axis base point
zaxis	axis direction vector
srtang	start angle (degrees)
endang	end angle (degrees)
return	name of created surface

Notes:

- The specified surface name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- Positions may be specified explicitly or using names of prescribed points
- For example usage, refer to `ic_geo_cre_srf_offset_edge.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_body_lower_entities *bdy*

Given a body name, *bdy*, it returns the names of the surfaces, curves, and points belonging to the body. These are returned in the form of argument pairs where the first name is the entity type and the second name is the entity name.

ic_geo_cre_geom_plot3d *in_file fit_tol* [0.0001] *pnt_fam* [PNT] *pnt_prefix* [pnt] *crv_fam* [CRV] *crv_prefix* [crv] *srf_fam* [SRF] *srf_prefix* [srf]

Creates point and b-spline geometry from a Plot3d file.

fit_tol	approximation tolerance
pnt_fam	family for point entities
pnt_prefix	prefix for point names
crv_fam	family for curve entities
crv_prefix	prefix for curve names
srf_fam	family for surface entities
srf_prefix	prefix for surface names

Notes:

- If the function returns with the error status set, the result string will contain an error message.
- MED must be initialized by loading a tetin file or opening an empty part (see *ic_empty_tetin*).
- The approximation tolerance is relative. It will be scaled by the pointset radius to form an absolute tolerance. It has a default value of 0.001.
- This routine is called by **ic_geo_cre_geom_input**

ic_geo_cre_srf_db_pnts *srfs*

Create the deboor points of a bspline surface.

srfs	list of surfaces
return	names of created points

ic_geo_cre_crv_db_pnts *crvs*

Creates the deboor points of a bspline curve.

crvs	list of curves
return	names of created points

ic_geo_read_off_file *fam name in_file*

Read an OFF file (native format for Geomview).

fam	family for new geometry
name	root name for new surfaces

in_file	input file

Notes:

- This function reads triangulated surfaces from the input file and creates unstructured surfaces in MED.
- Only minimal coverage of the OFF format has been implemented.
- For example usage, refer to `ic_geo_read_off_file.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_read_xyz_file *fam name in_file off_file mode* [""]

Reads and triangulate a list of XYZ points.

fam	family for new geometry
name	root name for new surfaces
in_file	input file
off_file	intermediate OFF file
mode	fast, tight

Notes:

- This function reads triangulated surfaces from the off file and creates unstructured surfaces in MED.
- Only minimal coverage of the OFF format has been implemented.
- For example usage, refer to `ic_geo_read_xyz_file.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.
- *mode* can be used to specify fast execution or watertight model.

ic_geo_crv_is_arc *crvs tol* [-1]

Determine whether one or more curves are circular.

crvs	list of one or more curves
tol	approximation tolerance
return	list of true/false flags

Notes:

- The tolerance defaults to **`0.001*arc_length`**
- For example usage, refer to `ic_geo_crv_is_arc.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_get_keypoints *dir border bb* [""]

Gets keypoints for the current geometry. *dir* is 0, 1, or 2. If *border* is non zero, then add some slack.

ic_geo_reverse_crv *crvs*

Reverses the orientation of one or more curves.

crvs	list of one or more curves
return	error message on failure

For example usage, refer to `ic_geo_reverse_crv.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_cre_edge_concat *crvs require_topo* [0]

Merges one or more curves and associated topology.

crvs	list of curves to be joined
require_topo	fail if topology will not merge
return	name of created curve

Notes:

- The specified curve name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- The list *crvs* contains curve *curve names*.
- If *require_topo* is set, the utility will fail if the associated topology cannot be merged. If *require_topo* is not set, the utility will create a new curve and preserve the original edges if the associated topology cannot be updated.
- For example usage, refer to `ic_geo_cre_edge_concat.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_create_histogram_box *min max lblList*

Make a histogram box.

min	min pt of the box
max	max pt of the box
lblList	list of labels

Notes:

- Each label in *lblList* is a list of {lbl_text, lbl_pt, ispt}.

- If $*ispt*==1, a point will be drawn as well as the label.

ic_geo_build_topo_on_srfs *srfs crvs* [""] *tol* [-1] *trim_srfs* [0] *concat_crvs* [0] *quiet* [0]

Builds topology on a list of surfaces.

srfs	surface(s)
crvs	optional curve(s)
tol	merge tolerance
trim_srfs	1 -- trim surfaces; 0 otherwise
concat_crvs	1 -- concatenate edges that join tangentially; 0 otherwise
quiet	1 -- suppress chatter; 0 otherwise

Notes:

- Merge tolerance will be determined from the bounding box of the surfaces.
- The merge tolerance defaults to **`0.0001 * (min surface radius)`**.

ic_geo_contact_surfaces *surfaces distance* [0] *family* [""] *debug* [0]

Search for contact surfaces.

surfaces	list of surface names
distance	maximum distance
family	family name for contact surfaces
return	list of surface pairs

ic_geo_map_tetin_sizes *tetin what* [0]

Map parameter data from a tetin file to the current model.

tetin	input tetin file
what	== 0 - map all possible data
	bit0 - map global parameters
	bit1 - map family parameters
	bit2 - map prescribed points data
	bit3 - map curve data
	bit4 - map surface data

For example usage, refer to `ic_geo_map_tetin_sizes.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_surface_thickness *surfaces order thickness* [""]

Set (or get) the thickness of surfaces.

surfaces	list of surface names
order	order of thickness approximation
thickness	order * order thickness values

ic_geo_srf_in_srf_fam_set *srf fams*

Determines whether a surface is within a volume bounded by one or more surface families.

srf	test surface
fams	list of families

ic_geo_cre_srf_over_holes *fam srfs*

Closes planar holes in a collection of surfaces.

srf	test surface
fams	list of families

ic_geo_subset_exists *name*

Checks if a geometry subset exists.

ic_geo_subset_copy *oldname newname*

Copies the geometry from one subset to another.

ic_geo_subset_clear *name*

Clears out everything from a subset.

ic_geo_subset_unused_name *pref* [subset]

Returns an unused geometry subset name with the given prefix Note that this gives names unique for both geometry and mesh.

ic_geo_subset_delete *name*

Deletes a geometry subset.

ic_geo_subset_visible *name vis*

Makes a geometry subset visible (or not).

ic_geo_subset_list_families *name*

Lists all the families that are represented in the named subset.

ic_geo_subset_list *pat* [*]

Lists all existing geometry subset names.

ic_geo_subset_add_items *name items*

Adds items to a geometry subset. If the subset does not exist, it will be created. The *items* argument is a list of type/name pairs. The types can be one of point, curve, surface, material, density, loop, body, shell, lump, solid, and the names specify the desired object of that type.

ic_geo_subset_remove_items *name items*

Removes items from a geometry subset. The items list is the same as for **ic_geo_subset_add_items**.

ic_geo_subset_handle_bc_changes *items subset add*

Adds or removes bc icons and groups based on addition or removal of objects in subsets.

ic_geo_subset_get_items *name*

Gets items in a geometry subset. This returns a list of type/name pairs.

ic_geo_subset_bbox *name*

Returns the bounding box of all geometry in a named subset.

ic_geo_subset_add_layer *name all feat_angle bound_mode*

Adds one or more layers to a geom subset. Arguments are the same as **ic_geo_flood_fill**.

ic_geo_subset_remove_layer *name*

Removes one or more layers from a geometry subset.

ic_geo_subset_names_to_parts *name*

Move the contents of all subsets to a part (name of subset)

ic_geo_get_srf_edges *srf*

Get any curves associated as edges to a surface.

ic_geo_get_vert_edges *pnt*

Get any curves associated as edges to a vertex.

ic_geo_calc_bisector_pnt *pnt1 pnt2 pnt3 len inverse*

Calculates node lying on the bisector of the angle formed by nodes *pnt1*, *pnt2*, *pnt3* in distance delta. If inverse: calculate the reverse bisector.

ic_geo_cre_srf_simple_trim *families names srf crvs*

Trims a surface using a simple contour.

families	list of 2 family names
names	list of 2 surface names

srf	surface to trim
crvs	list of trim curves

Notes:

- The specified surface names may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.

ic_geo_set_simplification_level *pixels*

Sets the simplification level in pixels (surfaces smaller than this size will be drawn as a box. 0 disables simplification. A value of -1 just returns the level.

ic_surface_thickness_check *names newfam* [""] *return_unassigned* [0]

Checks for zero thickness surfaces and assigns to new family. *newfam* is family name for surfaces with no thickness. *return_unassigned* is option to return unassigned surfaces without changing family. Default (0) is disabled.

ic_geo_close_contour *crvs srf*

Closes up a contour prior to trimming.

crvs	list of curves
srf	surface to trim

Notes:

- The specified surface names may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.

ic_geo_find_srf_prc_pnt *srf pnt vec*

Finds parameter of pierce point on surface.

srf	name of surface
pnt	test point
vec	pierce direction
return	uv pair

Notes:

- If the function returns with the error status set, the result string will contain an error message.
- The uv coordinates will be unitized.

- For example usage, refer to `ic_geo_find_srf_prc_pnt.tcl` in the Ansys installation directory under **`v212/icemcfd/Samples/ProgrammersGuide/med_test`**.

ic_geo_get_dormant *type only_if_visible* [""]

Returns list of dormant points or curves.

ic_geo_get_dormant_entity *type name*

Determines whether an entity is dormant Only points and curves can be dormant synchronized pickable and visible. Used in bounding box.

ic_get_facets *type list*

Returns list of faceted surfaces.

ic_geo_filter_curves *angle fams*

Returns a list of essential curves. A curve is "essential" when it bounds two surfaces which meet at an angle (measured by surface normals) exceeding a threshold angle. The function identifies the essential curves in the specified families.

angle	threshold angle in degrees
fams	names of families to search

ic_geo_cre_bridge_crv *fam name crv1 crv2 end1* [0] *end2* [0] *mag1* [0.3] *mag2* [0.3]

Creates a bridge curve between two curves.

fam	family of created curve
name	name of created curve
crv1	curve 1 name
crv2	curve 2 name
end1	curve 1 end indicator
end2	curve 2 end indicator
mag1	magnitude of start vector
mag2	magnitude of end vector
return	name of curve vertex

Notes:

- The curve end parameters, *end1*, *end2*, take the following values:

point closest to other crv	0

start point	1
end point	2

- 0 < mag1, mag2 < 1

ic_geo_cre_pln_crv *fam name crv base nrm*

Creates the projection of a curve onto a plane.

fam	family of created curve
name	name of created curve
crv1	curve name
return	name of created curve

ic_geo_pln_n_pnts *pnts*

Finds the least square plane through three or more points.

pnts	list of 3 or more points
return	{base} {normal} of plane

ic_geo_sub_get_numbers_by_names *type ent_names*

Returns each entity number (recognized in the batch interpreter) associated with each entity name in *ent_names*. *ent_names* can be a list but they must all be the same type defined by *type*. In this case, the return is a list of numbers in the same order as the entity names were given.

ic_geo_get_pnt_marked *name*

Determines whether a point is marked.

ic_geo_set_pnt_marked *name set_to*

Sets the marked flag on a point.

ic_geo_get_all_marked_pnts

Returns a list of all marked points.

ic_geo_add_embedded_crv *srf crvs*

Embeds a curve into a surface.

srf	surface

crvs	list of curves

Note:

No checks are performed to determine whether the curves are on the surface.

ic_geo_add_embedded_pnt *srf pnts*

Embeds a point into a surface.

srf	surface
pnts	list of points

Note:

No checks are performed to determine whether the points are on the surface.

ic_geo_is_crv_on_srf *crv srf tol* [-1]

Checks if a curve is on a surface.

crv	curve
srf	surface
tol	tolerance

Note:

If a negative value is passed for the tolerance, the utility will use an internally computed tolerance.

ic_geo_register_crv *crv_name new_fam*

Register a curve (this is used by Ansys TurboGrid)

crv_name	name of created curve

ic_geo_cre_midline_crv *crvs* [""] *toldebug* [0] *crvs2* [""] *fam* [""]

Creates a new curve by midlining two existing curves.

crvs	list of 2 curves
tol	tolerance
family	family containing curve

return	name of created midline curve

Notes:

- The specified curve name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- The list *crvs* contains 2 curves or curves of first set *curve names*.
- The list *crvs2* contains curves of second set *curve names*.

ic_geo_get_points_from_curves *curves*

This finds all the points attached to a list of curves.

ic_geo_test_cmd *vec pnt crvs srfs*

Test routine.

ic_geo_cre_crv_ell *family name center srt_pnt next_pnt srtang* [0] *endang* [360]

Creates a bspline ellipse from center, start point, second point.

family	family containing curve
name	name of created curve
center	ellipse center
srt_pnt	crv point on ellipse major axis
next_pnt	crv point not on major axis
srtang	start angle (degrees)
endang	end angle (degrees)
return	name of created curve

Notes:

- The specified curve name may be modified to resolve name collisions.
- If the function returns with the error status set, the result string will contain an error message.
- If *endang* < *srtang* or (*endang* - *srtang*) > 360, the angle will be adjusted by adding 360 increments.
- Positions may be specified explicitly or using names of prescribed points

ic_geo_improve_edge *crvs*

Improves one or more edges.

crvs	list of one or more curves

ic_geo_just_do_it *srfs*

Surface test routine.

srfs	list of one or more surfaces

ic_geo_get_prism_families

Returns the list of families for prism meshing.

ic_geo_set_prism_families *prism_fams excl* [1]

Sets the list of families for prism meshing. If excl==1, then any previous prism families are reset.

ic_geo_get_prism_family_params *fam*

Returns the prism meshing parameters for a family.

ic_geo_set_prism_family_params *fam args*

Sets the prism meshing parameters for a family.

ic_geo_create_tglib_sfbgrid *args*

Reads a TGLib size function background grid.

Usage: **ic_hex_create_tglib_size_functions**-*tglib_sfbgrid_file*`fname`-*remove_existing*-*create_callbacks*

e.g.,: **`eval ic_hex_create_tglib_size_functions -tglib_sfbgrid_file ./example.sf -remove_existing`**

Argument Descriptions:

-tglib_sfbgrid_file	full path to a tglib size function background grid file

Optional arguments:

-remove_existing	An existing size function background grid will be removed
-create_callbacks	Callback functions will be set for the usage of the size functions in e.g. the paver

ic_vcalc *op* [""] *args*

Calculates basic "vector" functions using triplets and/or point name. Returns empty string if not able to calculate result. The function (op) followed by one or more args (p1, p2, p3 or const), where p1, p2, p3 are triplets or point names const is integer or float scalar is shown:

nrm p1	normalized vector
dir p1 p2	normalized vector from p1 to p2
len p1	length of vector
distance p1 p2	distance between p1 and p2

sum p1 p2	vector sum
diff p1 p2	vector diff
smult p2 const	vector scaled
sum_smult p1 p2 const	vector sum scaled
dot p1 p2	dot product p1*p2
mult p1 p2	cross product p1×p2
product p1 p2	cross product p1×p2
angle p1 p2 p3	angle between p1-p2 and p1-p3 lines
angle p1 p2	angle between 000-p1 and 000-p2
vec_norm p1	some vector normal to p1

ic_highlight *args*

Temporary change color and/or "width" of points, curves, surfaces

- **color** white *or* red *or* green. Default color: family color.
- **width** 1 *or* 2 *or* 3. Default width: 1.
- **names** list of names. If list is empty, default settings will be restored

ic_vset *args*

ic_vset options/parameters : return ic_vset -names : all names* ic_vset -names type : names of that type ic_vset -method : all methods ic_vset -method type : methods for that type ic_vset -method type method : format of the method of the type ic_vset -method name : method for that name (if not default) ic_vset -database : all names of databases with items number ic_vset -database dname : make selected database current ic_vset name : empty line if not defined**, or value (if any) ic_vset name def : for variable set new definition ic_vset -type name : type of entity ic_vset -def name : definition of entity ic_vset -info name : detailed info on entity ic_vset -settings debug : return current debug level ic_vset -settings debug value : set current debug level ic_vset -settings med_pts : current med points usage option ic_vset -settings med_pts value : set current med points usage option ic_vset -settings interrupt value : set 0/1 interrupt design creation (ic_vcreate) ic_vset -vec - vec.expr. : calculate ï¿½anonymousï¿½ vector expression ic_vset -con - con.expr. : calculate ï¿½anonymousï¿½ constraint expression ic_vset -expression : calculate expression without database modification ic_vset -delete name : delete entity ic_vset -delete all : delete all entities in active database ic_vset : (without parameters) return last result or last reason for error** * options may be abbreviated to 3 or more characters: -nam, -met, -dat, -setï¿½ ** most commands return empty line on invalid input

ic_vdefine name type method_name* definition

ic_vfile read filename : read Vid file into current database ic_vfile write filename : write current database into Vid file

if {$npts == 2} {set pts [list $pts]}

ic_curve *method part name def args*

Create a bspline arc.

Usage: From a center point and two points: **ic_curvearc_ctr_rad** PART_NAME NEW_CURVE_NAME {CENTER_POINT POINT_1 POINT_2 0.0 "" "" 0}

From a center point and two points and radius: **ic_curvearc_ctr_rad** PART_NAME NEW_CURVE_NAME {CENTER_POINT POINT_1 POINT_2 RADIUS "" "" 0} Note: in case of a radius of 0.0, the arc radius will be calculated from the distance between CENTER_POINT and POINT_1

From start/end points: **ic_curvearc_ctr_rad** PART_NAME NEW_CURVE_NAME {POINT_1 POINT_2 POINT_3 0.0 "" "" 1}

From start/end points and radius: **ic_curvearc_ctr_rad** PART_NAME NEW_CURVE_NAME {POINT_1 POINT_2 POINT_3 RADIUS "" "" 1}

ic_geo_get_crv_data_at_par *par crv*

Returns the curve data at a parameter.

par	curve parameter
crv	curve name
return	list of 4 triplets: location, and 3 normalized vectors — tangent, direction to curve center, direction normal to curve plane
return	empty line if curve does not exist

Notes:

- For faceted curves result is parametric approximation of the curve.
- Direction to center and normal plane may be {0 0 0} if not defined, e.g. for line.

Blocking Commands

All **ic_hex** commands are used as a simple string without argument grouping. If an argument is grouped with curly braces, quotes, or square brackets, an *eval* should be placed before the command to remove one level of grouping.

Wherever an index range is needed as an argument, only specify the ranges which are limiting indices, not fully displayed indices. For example, for a box with max indices of 3 in i, j, and k (including VORFN), if only j and k are constrained, this argument may look like this: 0:2,3 2:1,3. The 1 index is not specified because it is not limited.

Use *[range_string]* to return the current active index range.

Wherever a list of volume parts is needed as an argument, *[volume_activename_string]* can be substituted for the current active parts that contain blocks. For example, the following command returns all visible vertex numbers:

```
eval ic_hex_select_nodes 1 all [range_string] m [volume_activename_string]
node_numbers
```

ic_hex_check_licensing

Checks if hexa licensing is enabled.

ic_hex_release_licensing

Check back hexa licensing.

ic_hex_init

Initialize the hexlib and graphics adapter. Usage: **ic_hex_init**

ic_hex_ensure_is_loaded

Returns 1 if the call to the function loads the hex library. Usage: **ic_hex_ensure_is_loaded**

ic_hex_dladaptobj_is_loaded

Returns 1 if hex graphics adapter is initialized. Usage: **ic_hex_dladaptobj_is_loaded**

ic_hex_remove_dladaptobj *args*

Remove hex graphics object if it is initialized. Usage: **ic_hex_remove_dladaptobj**

ic_hex_update_dladaptobj *args*

Updates *dladaptobj*.

ic_hex_is_modified *onoff* [""]

Checks to see if blocking has been modified, optional argument *onoff* 0/1 sets the blocking as not modified or modified.

ic_hex_curve_radius *args*

Returns the radius of the curve. Restrictions: Valid only for periodic models. Usage: **ic_hex_curve_radius***curve_name*

ic_hex_curve_length *args*

Returns the arc length of the curve. Usage: **ic_hex_curve_length***curve_name*

ic_hex_point_distance *args*

Returns the distance between the two prescribed points. Usage: **ic_hex_point_distance***p1p2*

ic_hex_error_messages *args*

Enables/disables the display of error messages. Usage: **ic_hex_error_messages** on|off|1|0

ic_hex_reference_blocking *args*

Usage:**ic_hex_reference_blocking***ref_blocking*

ic_hex_set_ref_uns_mesh *args*

Usage: **ic_hex_set_ref_uns_mesh***domain_file*

ic_hex_update_ref_uns_mesh *args*

Usage: **ic_hex_update_ref_uns_mesh**

ic_hex_set_uns_face_mesh *args*

Usage: **ic_hex_uns_face_mesh***uns_face*

ic_hex_surface_blocking *args*

Initializes a 2D block from every surface. Requires build topology first.

Usage: **ic_hex_surface_blocking***materialblock_type-min_edgevalue*

Argument Descriptions:

material	the blocks will belong to this part. *-inherited* will put each block into the part of the surface it came from
block_type	determines the unstructured and mapped blocks. Can be *-mapped*, *-mixed*, or *-unstruct*.
-min_edge	merge block if any edge of a block is less than the following number. Only applies if the *-respect_non_dormant* argument is not added
value	value for **-min_edge**

Additional arguments:

-respect_non_dormant	blocks will not be merged if this argument is appended
-surfaces	only initializes the listed surfaces. This is followed by a list of surface names.

ic_hex_multizone_blocking *args*

Initializes a 3D blocking. Requires build topology first. Geometry must contain a solid (from imported geometry) or a body.

Usage: **ic_hex_multizone_blocking***-inherited-block_type-min_edgevalue1-uface_typevalue2-volfampart*

e.g.,: **`ic_hex_multizone_blocking -inherited -block_type -min_edge value1 -automatic_sweep -uface_type value2 -volfam part`**

Argument descriptions:

-block_type can be **robust** or **swept**.

-min_edge merge blocks if any edge of a block is less than the **value1** number. Only applies if the **-respect_non_dormant** argument is not added.

value1 Value for **-min_edge**.

-uface_type can be one of **several_tris** or **all_tris** or **all_quads**.

value2 Value for **-uface_type**.

Additional arguments:

-respect_non_dormant	Blocks will not be merged if this argument is appended.
-automatic_sweep	In case of block_type **swept** required.
-swept_surfaces	The surfaces for the swept blocks. This is followed by a list of surface names.

ic_hex_virtual_topo_blocking *args*

Creates 2D surface blocking for AutoVT

ic_hex_twod_to_threed *args*

Fills in 2D surface blocking with volume blocks. Usage: **ic_hex_twod_to_threed***part[-basic]*

part is the part that the new volume blocks will be created in.

If *-basic* is appended, the volume blocks will all be unstructured blocks if it cannot figure out a simple structured fill. If *-basic* if omitted, it will try to break up the domain in order to get as many structured blocks as possible.

The basic fill is the most robust and most used. The 2D blocking must form a closed volume of surface blocks.

ic_hex_make_edge_smooth *args*

Usage: **ic_hex_make_edge_smooth***n1n2*

ic_hex_reset_ogrid_orthogonality *args*

Resets the Ogrid orthogonality between a vertex and the orthogonal vertex along Ogrid direction.

Usage: **ic_hex_reset_ogrid_orthogonality***n1n2[parts]*

ic_hex_blank_blocks *args*

Blanks blocks. Usage: **ic_hex_blank_blocks** [marked] [reverse]

ic_hex_blank_selected_blocks *use_output use_blank blocks*

Blank selected blocks. Usage: **ic_hex_blank_selected_blocks***use_output use_blank blocks*. Set *use_output* to "1" if **Pre-Mesh** → **Output Blocks** is set. Set *use_blank* to "1" if **Blocks** → **Blanking**is set. *blocks* is a list of block numbers.

ic_hex_unblank_blocks *args*

Unblanks blocks. Usage: **ic_hex_unblank_blocks** [marked] [reverse]

ic_hex_check_face_radii *args*

Usage: **ic_hex_check_face_radii***n1n2n3n4*

ic_hex_update_node_indices *args*

Updates the node indices. Usage: **ic_hex_update_node_indices**

ic_hex_update_node_locations *args*

Updates the node locations based on geometry tags and projection of vertices. Usage: **ic_hex_update_node_locations** [parts]

ic_hex_remove_unused_grid_lines *args*

Removes unused grid lines from the blocking

Usage: **ic_hex_remove_unused_grid_lines** [-only_one] [-version]

ic_hex_set_edge_tangents *args*

Sets the tangents on an edge Usage: **ic_hex_set_edge_tangents***n1n2tan1_infotan2_info*

Where: *tan1_info* would be one of following: tan1 ortho fac1 tangency_factor tan1 tangency_point_xyz fac1 tangency_factor "" and *tan2_info* would be one of following: tan2 ortho fac2 tangency_factor tan2 tangency_point_xyz fac2 tangency_factor "" e.g. set n1 22 set n2 65 eval ic_hex_set_edge_tangents $n1 $n2 tan1 ortho fac1 0.33 tan2 0 0 1 fac2 0.33

ic_hex_smooth_edges_at_vertex *args*

Makes the edges coming into a vertex smooth. Usage: **ic_hex_smooth_edges_at_vertex***node_numfamilies*

node_num could be replaced by *ijk_range* for example,
```
set n 26 eval
ic_hex_smooth_edges_at_vertex $n [volume_activename_string]
```

ic_hex_edge_node_dim *args*

Returns the dimension and index between two vertices. Usage: **ic_hex_edge_node_dim***n1n2*

ic_hex_get_edge_node *args*

Returns the other node of the edge dimension. Usage: **ic_hex_get_edge_node***ijk dim* for example, **`ic_hex_get_edge_node " 1 2 1 " 0`**

ic_hex_point_location *args*

Returns the location of a prescribed point. Usage: **ic_hex_point_location***name*

ic_hex_get_surface_family *args*

Returns the family name of a surface. Usage: **ic_hex_get_surface_family***surface_name*

ic_hex_define_family *args*

Defines a family by a name. Usage: **ic_hex_define_family***name*

ic_hex_set_family_projection *args*

Allows you to enable/disable projection to a part.

Usage: **ic_hex_set_family_projection***partname0|1*

for example, **`ic_hex_set_family_projection SOLID 0`** indicates that projection to the part SOLID has been disabled. Similarly, **`ic_hex_set_family_projection FLUID 1`** indicates that projection to the part FLUID has been enabled.

ic_hex_auto_split_edge *args*

Splits an edge into as many points as it will be meshed. Usage: **ic_hex_auto_split_edge** {n1 n2} | {-all [-spline] [dim : low , hi] [fam1 fam2 ...]}

ic_hex_split_edge *args*

Splits an edge at the specified location. Usage: **ic_hex_split_edge** {n1 n2} {n} {loc}

for example, **`ic_hex_split_edge 21 25 0 1.7 8.5 0`** implies that edge 21–25 will be split at the location (1.7, 8.5, 0). The value *n* which is **`0`** in this example indicates that this is the first split between the vertices 21 (start vertex) and 25 (end vertex). Alternatively, a value of **`1`** would indicate that the split is the second between the specified vertices, and so on.

Tip:

To use an existing point to indicate the location to be used, you can use the following in the replay script:

```
set name pnt.00 ;# point must exist
set loc [ic_geo_get_point_location $name]
set x [lindex $loc 0]
set y [lindex $loc 1]
set z [lindex $loc 2]
ic_hex_split_edge 21 25 0 $x $y $z
```

ic_hex_undo_major_start *args*

Puts a major start in the undo log and save model. Usage: **ic_hex_undo_major_start***text*

ic_hex_undo_major_end *args*

Puts a major end in the undo log. Usage: **ic_hex_undo_major_end***text*

ic_hex_add_boco *args*

Adds a boundary condition to a family. Usage: **ic_hex_add_boco***family* [value1 value2 ...]

ic_hex_get_block_family *args*

Gets the family (part) of a block. Usage: **ic_hex_get_block_family***block_num*

ic_hex_read_domains *args*

Reads the node locations from the domains directory. Usage: **ic_hex_read_domains***m1m2* ...

ic_hex_reverse_dimension *args*

Reverses the global orientation of a dimension. Usage: **ic_hex_reverse_dimension** [dim]

ic_hex_write_family_boco *args*

Writes the family boco file. Usage: **ic_hex_write_family_boco** [filename]

ic_hex_clear_bocos *args*

Clears the bocos on some or all families Usage: **ic_hex_clear_bocos** [fam1 ...]

ic_hex_align_superblocks *args*

Aligns blocks with a given superblock. Usage: **ic_hex_align_superblocks***block_num*

ic_hex_vertex_ijk *args*

Returns the i j k of a vertex. Usage: **ic_hex_vertex_ijk***vertex_num*

ic_hex_vertex_number *args*

Returns the vertex number from the vertex index.

args is a vertex index in format { i j k etc }. There must be spaces on both sides of each number.

ic_hex_get_block_vertices *args*

Returns the 8 corner vertices of the block. Usage: **ic_hex_get_block_vertices***block_num*

ic_hex_get_vertex_blocks *args*

Returns block numbers of all blocks attached to the vertex. Usage: **ic_hex_get_vertex_blocks***vertex_num*

ic_hex_get_edge_blocks *args*

Returns block numbers of all blocks attached to the edge. An edge is defined by its end vertex numbers.

Usage: **ic_hex_get_edge_blocks***vertex_num1vertex_num2*

Usage: **ic_hex_get_edge_blocks***{ vertex_num1 vertex_num2 }*

ic_hex_get_face_blocks *args*

Returns block numbers of attached blocks that contain the specified face. A face is defined by its 4 corner vertex numbers.

Usage: **ic_hex_get_face_blocks***vertex_num1vertex_num2vertex_num3vertex_num4*

Usage: **ic_hex_get_face_blocks***{ vertex_num1 vertex_num2 vertex_num3 vertex_num4 }*

ic_hex_get_vertex_faces *args*

Returns faces attached to the specified vertex. A face is defined by its 4 corner vertex numbers.

Usage: **ic_hex_get_vertex_faces***vertex_num*

ic_hex_get_edge_faces *args*

Returns faces attached to the specified edge. A face is defined by its 4 corner vertex numbers. An edge is defined by its end vertex numbers.

Usage: **ic_hex_get_edge_faces***vertex_num1vertex_num2*

ic_hex_get_vertex_edges *args*

Returns edges attached to the specified vertex. An edge is defined by its end vertex numbers.

Usage: **ic_hex_get_vertex_edges***vertex_num*

ic_hex_uncollapse_edge *args*

Uncollapses an edge of a degenerate block. Usage: **ic_hex_uncollapse_edge***vertex_num*

ic_hex_link_shape *args*

Links the shape of one edge to another. Usage:
ic_hex_link_shape*tar_vert1tar_vert2src_vert1src_vert2factor*

tar_vert1 and *tar_vert2* are the end vertices of the target edge. *src_vert1* and *src_vert2* are the end vertices of the source edge. **factor** is a multiple of the curvature. 1 will produce the exact same shape. A decimal will produce less curvature, and a number greater than one will produce more curvature.

ic_hex_link_shape_dimension *args*

Links the shape of all edges across the specified index dimension

Usage: **ic_hex_link_shape_dimension***dimvertex1vertex2factorindex_rangeparts*

Argument descriptions:

dim	index dimension where i=0, j=1, k=2, O3=3, etc..
vertex1	vertex number belonging to one of the source edges
vertex2	vertex number belonging to one of the target edges
factor	multiplication factor of the shape curvature. "1" will produce the same curvature.
index_range	operates only on this *index_range*. Use [range_string] to return the current active index range.
parts	the list of parts that contain blocks. Only edges belonging to these blocks will be shaped. Use [volume_activename_string] to return all the active part names containing blocks.

ic_hex_topo_vertex_number*args*

Returns the topological number of a face. Usage: **ic_hex_topo_vertex_number***n1*

ic_hex_topo_edge_number *args*

Returns the topological number of an edge. Usage: **ic_hex_topo_edge_number***[output_blocks]n1n2*

ic_hex_list_edge_constraints *args*

Returns the constraints set on an edge. Usage: **ic_hex_list_edge_constraints***n1n2*

ic_hex_topo_face_number *args*

Returns the topological number of a face. Usage: **ic_hex_topo_face_number** [output_blocks] *n1n2*

ic_hex_topo_face_number_list *args*

Returns the topological numbers of a list of faces. Usage: **ic_hex_topo_face_number_list** [output_blocks] "*n1n2n1n2* ..."

ic_hex_get_super_faces_topo *args*

Returns the topological numbers of all existing faces. Usage: **ic_hex_get_super_faces_topo** [output_blocks]

ic_hex_change_mouse_buttons *args*

Changes mouse buttons for Hexa. Usage: **ic_hex_change_mouse_buttons***rotatetranslatescalescale_dir*

ic_hex_global_smooth *args*

Runs the global hex smoother on the blocking. Usage: **ic_hex_global_smooth***families* [fam1 fam2 fam3 ...]

ic_hex_edge_ijk *args*

Usage: **ic_hex_edge_ijk***topo_num*

ic_hex_gls_init *args*

Usage: **ic_hex_gls_init***name*

ic_hex_display_ijk_region *args*

Display faces between different materials.

Usage: **ic_hex_display_ijk_region** [topo_dom_no dom] [ijk_reg n1 n2 n3 n4 n5 n6] [solid]

ic_hex_delete_blocks *args*

Deletes blocks permanently from the blocking.

Usage: **ic_hex_delete_blocks** {ijk1 ijk2} | marked | {numbers n1 n2 ...} | {blanked fam1 fam2 ...} [output_blocks]

ic_hex_mesh_dimension *args*

Gets the dimension of the blocking (2=2D Blocking or 3=3D Blocking). Usage: **ic_hex_mesh_dimension**

ic_hex_dim_to_mesh *args*

Gets the meshed dimension of the blocking (1,2,2.5 or 3). Usage: **ic_hex_dim_to_mesh**

ic_hex_create_block *args*

Creates a new block.

Usage: **ic_hex_create_block***part_namen1n2n3n4n5n6n7n8active_parts*

Usage: **ic_hex_create_block***part_namen1n2n3n4active_parts* (for a 2D blocking)

e.g.,: **`eval ic_hex_create_block SOLID 134 135 128 129 110 111 104 105 [volume_activename_string]`**

For the two opposing rectangular faces of the 3D block, n2 directly opposes n1, n4 directly opposes n3, n6 directly opposes n5, and n8 directly opposes n7

Any of n1 - n8 can be replaced with an {x y z} location

ic_hex_create_sheet *args*

Creates a 2D sheet in 3D Blocking.

Usage: same as **ic_hex_create_block**

ic_hex_create_unstruct_sheet *args*

Creates a 2D unstructured sheet in 3D Blocking.

ic_hex_merge_sheet_with_unstruct *args*

Merges the sheet block inside a free block.

Usage: **ic_hex_merge_sheet_with_unstruct***free_blocksheet_block*

ic_hex_create_block_primitive *args*

Creates a block primitive. Usage: **ic_hex_create_block_primitive***typefamily* c1 c2 [c3] p1 p2 [p3] s1 s2 [s3]

ic_hex_create_yblock *args*

Creates a new Y-block.

Usage: **ic_hex_create_yblock***part_namen1n2n3n4n5n6active_parts*

e.g.,: **`eval ic_hex_create_yblock SOLID 134 135 110 128 129 104 [volume_act-ivename_string]`**

The first 3 vertices belong to the triangular face of one side, and the last 3 vertices are for the triangular faces of the opposite side

Any of n1 - n6 can be replaced with an {x y z} location

ic_hex_create_degen_block *args*

Creates a new degenerate block.

Usage: **ic_hex_create_degen_block***part_namen1n2n3n4n5n6active_parts*

e.g.,: **`eval ic_hex_create_degen_block SOLID 134 135 110 128 129 104 [volume_activename_string]`**

The mesh will converge to n1 and n4. Any of n1 - n6 can be replaced with an {x y z} location.

ic_hex_create_unstruct_block *args*

Creates a swept block.

Usage: **ic_hex_create_unstruct_block***part_namen1n2n3n4n5n6active_parts*

Usage: **ic_hex_create_unstruct_block***part_namen1n2n3* ... *nn* active_parts (for a 2D blocking)

e.g.,: **`eval ic_hex_create_unstruct_block SOLID 134 135 110 128 129 104 [volume_activename_string]`**

For the two opposing triangular faces of the block, n2 directly opposes n1, n4 directly opposes n3, and n6 directly opposes n5. If a block is converted to unstructured, rather than created, it is not limited to a 6-vertex block in shape.

ic_hex_split_unstruct_block *args*

Split a free block face.

Usage: **ic_hex_split_unstruct_block***vertex1vertex2*

vertex1 and vertex2 must be corner vertices of the face to split.

ic_hex_split_unstruct_block3d *args*

Splits a 3D unstructured block into different volume regions.

Usage: **ic_hex_split_unstruct_block3d***ublock_numby_sheetssheet1sheet2...*

Usage: **ic_hex_split_unstruct_block3d***ublock_numloop1sheet1*

Usage: **ic_hex_split_unstruct_block3d***ublock_numloop1sheet1loop2sheet2*

ic_hex_select_blocks *args*

Returns block numbers in the range and parts specified.

Usage: **ic_hex_select_blocks** num display method index_range m parts

e.g.,: **`ic_hex_select_blocks 1 no_highlight all [range_string] m [volume_activename_string]`**

Argument descriptions:

num	usually 1
display	*no_highlight* to not highlight the block
method	*all*, or *inpoly* p1 p2 p3 p4, where each p is the xyz of a point
index_range	specifies the index range to limit selection. Use [*range_string*] to return the current active index range
m	indicates the following arguments are part names
parts	the list of parts that contain blocks. Use [*volume_activename_string*] to return all the active part names containing blocks

The following arguments may be appended:

-not_blanked	Normally, blanked blocks do not affect which vertices are returned. This option ensures vertices turned off due to blanked blocks are not returned.
super	
-keep_unstruct3d	

ic_hex_default_bunching_law *args*

Sets the default bunching law and ratio

Usage: **ic_hex_default_bunching_law***lawratio*

Returns without any argument the current bunching law.

ic_hex_merge_blocking *args*

Merge blocking topologies

Usage: **ic_hex_merge_blocking***topo1topo2parts-remove_unused_grid_lines-eps<tolerance>-separate_classes-version 110*

Where,

- *topo1* and *topo2* are the topologies to merge

- *parts* is the list of parts that contain blocks

  Use *[volume_activename_string]* to return all the active part names containing blocks.

- *tolerance* specifies a relative tolerance for merging the topologies

ic_hex_merge_face *args*

Merges faces.

Usage by face corners: **ic_hex_merge_face***[ijk of vertex_num1][ijk of vertex_num2]*

Usage by block faces: **ic_hex_merge_face***-facesface1face2*

ic_hex_print_topo_tree *args*

Returns the existing topology names.

ic_hex_extrude_blocks *args*

Converts a 2D blocking into a 3D blocking by translation or rotation.

Usage: **ic_hex_extrude_blocks***translatedir*

Usage: **ic_hex_extrude_blocks***rotatecenteraxisangle*

e.g.,: **`ic_hex_extrude_blocks rotate 0 0.05 0 1 0 0 45 -ncopies 1 -npoints 10 collapse_axis -extrude_points -extrude_curves`**

Argument descriptions:

dir	vector which describes direction and magnitude of translation
center	xyz location of center of rotation
axis	vector which gives the rotation axis direction
angle	angle of rotation in degrees

Additional arguments for the rotation method:

-ncopies	Each 3D wedge will be incrementally copied around the axis. A number must follow this argument. The first extrusion is included in the number. 4 copies of 90 degrees will produce a complete 360 degrees.
-npoints	The number of nodes on extruded edges in the circumferential direction. A number must follow this argument.
collapse_axis	Merges nodes at the axis. Choose this if vertices lie on the axis.
-extrude_points	The points of point-projected vertices (red vertices) will be extruded into curves.
-extrude_curves	The curves of curve-projected edges (green edges) will be extruded into surfaces.

ic_hex_switch_blocking *args*

Switch to another blocking topology

Usage: **ic_hex_switch_blocking***topo_name*

ic_hex_merge_blocks *args*

Merge two or more mapped or free blocks.

Usage 2D:

- **ic_hex_merge_blocks***block_numublock_num*
- **ic_hex_merge_blocks***ublock_num1ublock_num2*
- **ic_hex_merge_blocks***block_num1block_num2block_num3* ...

Usage 3D:

- **ic_hex_merge_blocks***block_num1block_num2block_num3* ...
- **ic_hex_merge_blocks***ublock_num1ublock_num2ublock_num3* ...

where *block_num* means **mapped** block and *ublock_num* means **free** block.

ic_hex_get_proj_curves *compcurve*

Returns the curve names that make up the composite curve.

ic_hex_find_comp_curve *curve*

Returns the composite curve name given any of the curve names that belong to it. Hexa composite curves are given the name of the first curve in the list of curve names that make up the composite curve.

ic_hex_delete_node_on_edge *args*

Remove nodes from faces in a free 3D block. Removal of node is not allowed for block corners.

Usage: **ic_hex_delete_node_on_edge***vertex_num*

ic_hex_set_mesh_params *args*

Sets the mesh params for the pre-mesh

Usage: **ic_hex_set_mesh_params***parts<fix_counts><fix_laws><curve_bunching>-version 110*

Where,

- *parts* is the list of parts that contain blocks.

  Use *[volume_activename_string]* to return all the active part names containing blocks
- *fix_counts*, if specified, changes the edge bunching law to the default (BiGeometric) geometry law
- *fix_laws*, if specified, adjusts the number of nodes on each edge based on the global surface or curve mesh size

- *curve_bunching*, if specified, transfers the advanced bunching parameters specified for curves to their associated edges

ic_hex_composite_curve *args*

Creates a composite curve from the list of curves. The composite curve name is the name of the first curve in the list.

Usage: **ic_hex_composite_curve***curve1curve2curve3* ...

ic_hex_mark_blocks *args*

Marks blocks to be later used in operations on specific blocks. These include block splitting, Ogrids, and transformation operations.

Usage: **ic_hex_mark_blocks***superblockblock_number*

Usage: **ic_hex_mark_blocks***unmark*

Only one *block_number* is allowed. Repeat the command for multiple block numbers.

The *unmark* version of the command clears all the marked blocks. This should always be done before marking new blocks.

ic_hex_select_nodes *args*

Returns vertex indices in the index range and block parts specified.

Usage: **ic_hex_select_nodes***nummethodindex_rangemparts*

e.g.,: `ic_hex_select_nodes 1 all 0:2,3 2:1,3 m [volume_activename_string]`

Argument descriptions:

num	1
method	*all*, or *inpoly* p1 p2 p3 p4, where each p is the xyz of a point
index_range	Specifies the index range to limit selection. Use [*range_string*] to return the current active index range
m	indicates the following arguments are part names
parts	the list of parts that contain blocks. Use [*volume_activename_string*] to return all the active part names containing blocks

The following arguments may be appended:

node_numbers	Return node numbers instead of indices
surface	Return only external vertices
-not_blanked	Normally, blanked blocks do not affect which vertices are returned. This option ensures vertices turned off due to blanked blocks are not returned.

ic_hex_select_edge*args*

Returns edge ID's in the range and parts specified

Usage: **ic_hex_select_edge***displaymethodindex_rangemaxnummparts*

e.g.,: **`ic_hex_select_edge no_highlight all [range_string] max 1 m [volume_activename_string] new_format edge_segment volume`**

Argument descriptions:

display	*no_highlight* to not highlight the edge
method	*al*, or *inpoly* p1 p2 p3 p4, where each p is the xyz of a point
index_range	Specifies the index range to limit selection. Use [*range_string*] to return the current active index range
max	the text "max" indicates the following number is the maximum amount to select
num	usually 1
m	indicates the following arguments are part names
parts	The list of parts that contain blocks. Use [*volume_activename_string*] to return all the active part names containing blocks

The following arguments may be appended:

new_format	
edge_segment	
volume	return internal volume edges also. Otherwise just return external edges.

ic_hex_display_edge_mesh *args*

Sets the visible index range for the blocking.

Usage: **ic_hex_display_edge_mesh***index_rangempartsvolume*

e.g.,: **`ic_hex_display_edge_mesh 1:2,3 m [volume_activename_string] volume`**

Specify only the ranges which are limiting indices, not fully displayed indices. For example, for a box with max indices of 3 in i, j, and k (including VORFN), if only j and k are constrained, this argument may look like this: 0:2,3 2:1,3. The 1 index is not specified because it is not limited.

To display the full index range, leave the argument empty. To return the current index range, use the command *range_string*.

ic_hex_select_super_faces *args*

Returns face ID's in the range and parts specified.

Usage: **ic_hex_select_super_faces***methodmaxnumindex_rangeparts*

Usage: **ic_hex_select_super_faces***methodmaxnumcornersv1v2index_rangeparts* e.g.,: `ic_hex_select_super_faces all max 1 [range_string] [volume_active-name_string]`

Argument descriptions:

method	*all*, or *inpoly* p1 p2 p3 p4, where each p is the xyz of a point
max	indicates the following number is the maximum amount to select
num	usually 1
corners	indicates the faces will be selected by the two corners method
v1, *v2*	vertex numbers for the two corners method. Diagonally opposed vertices.
index_range	Specifies the index range to limit selection. Use [*range_string*] to return the current active index range
parts	The list of parts that contain blocks. Use [*volume_activename_string*] to return all the active part names containing blocks

ic_hex_project_to_surface *args*

Move the nodes to surfaces, curves, and points

Usage: **ic_hex_project_to_surface***verticesindex_rangeparts<move_ogrid_nodes>*

Where,

- *vertices* is a list of vertices to project
- *index_range* sets the index_range on which the function operates. Use [range_string] to return the current active index range.
- *parts* is the list of parts that contain blocks
- *<move_ogrid_nodes>* optionally moves the ogrid nodes. Default is to not move the ogrid nodes.

ic_hex_ogrid *args*

Create ogrid

Usage: **ic_hex_ogrid***partsoffset<link_shape><-version 50>*

Where,

- *parts* is the list of parts that contain blocks
- *offset* specifies the height of the ogrid layer
- *link_shape*, if specified, causes all internal edges and faces of ogrid block to be shaped by the nearest, corresponding geometry.

ic_hex_list_block_families *args*

Returns only block parts

ic_hex_list_block_families_without *skip* [VORFN]

Returns all block parts except `part_name`.

Example usage: **ic_hex_list_block_families_without`part_name`**

ic_hex_list_surface_families *args*

Returns only blocking surface families

ic_hex_list_block_numbers *args*

Returns the list of super block numbers

ic_hex_non_empty_families *args*

Returns all non-empty block parts

ic_hex_save_blocking *args*

Saves the current blocking

Usage: **ic_hex_save_blocking***file<-sub_topo topo>*

ic_hex_split_grid *args*

Splits hexa blocks.

Usage: **ic_hex_split_grid***vertex1vertex2valuempartsflags*

Usage: **ic_hex_split_grid***indexdimvaluempartsflags*

Argument descriptions:

vertex1	vertex number at end of edge
vertex2	vertex number at other end of edge
value	split value • a decimal between 0 and 1 • *abs*:distance • point name • *curve*:curvename:parameter
m	indicates the following are part names
parts	parts to split through
flags	When set to marked, only the *marked* blocks are split. The command **ic_hex_mark_blocks** will mark blocks
index	index of vertex on the lower side of *dim* in format { i j k etc }
dim	dimension of edge to split where 0=i, 1=j, 2=k, 3=O3, etc...

ic_hex_restore_blocking *args*

Loads a blocking file

Usage: **ic_hex_restore_blocking***blocking_filename<-skip_read_surface_params>*

If specified, *-skip_read_surface_params* will ignore the size parameter from the tetin file. This is useful if you set other size parameters to the blocking and do not want them to be overwritten. Default is to transfer size parameters automatically to the blocking.

ic_hex_blocking_loaded *args*

Returns whether or not a blocking is loaded.

Usage: **ic_hex_blocking_loaded**

ic_hex_remove_edge_point *args*

Interactively remove edge split vertices

ic_hex_ratio_histogram *args*

Returns a list with ratio values of the selected quality criterion:

- Value 1 is the minimum value found.
- Value 2 is the maximum value found.
- The number of the next values depends on the number of bars specified. For each bar (range) it will return 4 values: number of elements, the lowest value, the highest value, and percent.

Usage: eval **ic_hex_ratio_histogram***number_of_barsparts* proj *projection_type* minval *minval_value* -type *criterion* maxval *maxval_valuenew_format*

Where,

- *number_of_bars* is the number of requested bars
- *parts* is a list of parts that contain blocks
- set *projection_type* to -1 (no projection), 0 (vertices), 1 (edges), or 2 (faces)
- *minval_value* sets the minimum value of the requested ratio range
- *criterion* specifies the quality criterion to be used
- *maxval_value* sets the maximum value of the requested ratio range

ic_hex_undo *args*

Obsolete

ic_hex_redo *args*

Obsolete

ic_hex_new_blocking *ents fam bbox* [""] *trf* [""] *version* [50]

Obsolete; use ic_hex_initialize_blocking

ic_hex_keypoint_blocking *args*

Initialize blocking from key points. Initialize a 3D blocking from entities. Initializing is required before any other blocking commands can be done.

Usage: **ic_hex_initialize_blocking***entspartoriented* {version 101}

ic_hex_convert_to_struct *args*

Convert a block to structured (mapped).

Usage:

- 2D or 3D blocking: **ic_hex_convert_to_struct***block_number-iterative*

  Example: `ic_hex_convert_to_struct 44 -iterative`

- 3D free or swept block face: **ic_hex_convert_to_struct***free_face_number-propagate*

  Example: `ic_hex_convert_to_struct { 32 - 33 u } -propagate`

for

Usage for

ic_hex_convert_to_swept *args*

Convert a block to swept or convert a free block free face to mapped.

Usage:

- for convert a block to swept: **ic_hex_convert_to_swept***block_face [-merge_parallel]*
  -merge_parallel will merge all adjacent mapped faces at the same index level.

- for convert a free block free face to mapped: **ic_hex_convert_to_swept***free_block_face*

Note:

only for 3D blocking

ic_hex_imprint_loop *args*

Imprint loops of edges from one face onto another free block (2D) or free face (3D).

Usage for 2D blocking: **ic_hex_imprint_loop***target_blockedge_loops*

Usage for 3D blocking: **ic_hex_imprint_loop***target_faceedge_loops*

ic_hex_imprint_loop_and_split_block3d *args*

Combines into a single operation the ability to imprint loops from one or more sets of edges onto a free face, then split the block between the pair(s) of loops.

Usage: **ic_hex_imprint_loop_and_split_block3d***free_faceedge_loops*

ic_hex_connect_edges *args*

Connect edges from one block onto another free block (2D only).

Usage for 2D blocking: **ic_hex_connect_edges***target_blockedges*

ic_hex_create_swept_block *args*

Create a swept block by merging two faces.

Usage: **ic_hex_create_swept_block***face1face2*

ic_hex_unstruct_face_type *args*

Sets the type of mesh for unstructured 2D blocks

Usage: **ic_hex_unstruct_face_type***type*

type can be *stl*, *all_tris*, *all_quads*, *several_tris*, or *one_tri*

ic_hex_print_block_type_info *args*

Returns the block type (free, mapped or swept).

Usage: **ic_hex_print_block_type_info***block_num*

ic_hex_create_named_selection_subset *args*

Creates a named selection subset.

Usage: **ic_hex_create_named_selection_subset***name*

name is applied to the subset.

ic_hex_subset_add_named_selection_items *args*

Add entities to a named selection subset.

Usage: **ic_hex_subset_add_named_selection_items***nametypenumber_of_entitiesentities*

Argument descriptions:

name	identifies the subset
type	can be *vertex*, *edge*, or *face*
number_of_entities	an integer value for the quantity of items to be added
entities	identifies the entities that are to be added using this type-dependent syntax: • If *vertex*, use the vertex number followed by 0 -1. For example *15 0 -1* • If *edge*, use the end vertices followed by -1. For example *3 15 -1*

	• If *face*, use the four corner vertex numbers. For example, *14 15 19 18* for a mapped face like { 14 15 19 18 } or *18 24 -1 -1* for a free face like { 18 - 24 u }.

Example: ic_hex_subset_add_named_selection_items *WALL face 4 14 19 15 18 15 18 16 17 12 13 16 17 12 13 14 19* will add **4face**s to the named selection subset called **WALL**. The four faces have corner vertices { 14 19 15 18 }, { 15 18 16 17 }, { 12 13 16 17 }, and { 12 13 14 19 }.

ic_hex_subset_remove_named_selection_items *args*

Remove entities from a named selection subset.

Usage: **ic_hex_subset_remove_named_selection_items***nametypenumber_of_entitiesentities*

Argument descriptions: See the table under ic_hex_subset_add_named_selection_items.

ic_hex_delete_named_selection_subset *args*

Delete a named selection subset.

Usage: **ic_hex_delete_named_selection_subset***name*

ic_hex_rename_named_selection_subset *args*

Rename a named selection subset.

Usage: **ic_hex_rename_named_selection_subset***namenew_name*

ic_hex_subset_named_selection *name*

Returns 1 if the named selection subset exists, 0 otherwise.

Usage: **ic_hex_subset_named_selection***name*

ic_hex_subset_list_named_selections

Returns a list of all named selection subsets.

Usage: **ic_hex_subset_list_named_selections**

ic_hex_mesh_free_surfaces *args*

Use Quad to mesh all of the surfaces with unstructured/free blocks.

ic_hex_get_edge_param *n1 n2 param*

Returns edge parameters.

Usage: **ic_hex_get_edge_param***vertex1vertex2parameter*

Argument descriptions:

vertex1	vertex number at end of edge

vertex2	vertex number at other end of edge
parameter	parameter value of edge to return • nodes (number of nodes) • len (edge length) • law (mesh law) • max (max space) • sp1a or sp2a (actual spacing 1 or 2) • sp2r or sp2r (requested spacing 1 or 2) • r1a or r2a (actual ratio 1 or 2) • r1r or r2r (requested ratio 1 or 2)

ic_hex_is_edge_linked *n1 n2*

Returns 1 if edge is linked to another edge, and 0 if not.

Usage: **ic_hex_is_edge_linked***edge_vertex1edge_vertex2*

ic_hex_match_and_link_edges *rn1 rn2 n1 n2*

Allows you to match and link spacing on edges.

Usage: **ic_hex_match_and_link_edges***from_edgeto_edge*

from_edge is the reference edge, *to_edge* is the target edge(s).

ic_hex_copy_edge_params *from to absolut* [0] *no_nodes* [0]

Allows you to copy the edge parameters from a reference edge to the selected edge(s) or to all parallel edges.

Usage: **ic_hex_copy_edge_params***from_edgeto_edgeabsolutno_nodes*

from_edge is the reference edge. *to_edge* could be a list of selected edges or the value *copy_to_parallel*. *absolut* indicates that the exact spacing will be copied (default 0). *no_nodes* determines whether the number of nodes will be copied (default 0).

ic_hex_ratio_check_minquality *which fams* [""] *range* [""] *project_type* [2] *full* [0]

Checks and returns the minimum quality ratio (in case of "Volume change" the maximum quality).

Usage: **ic_hex_ratio_check_minquality** "*criterion*" [*parts to check*] [*index range*] [*project_type 2*]

ic_hex_scale_mesh_size_with_spacings_gui *scale fams* [""] *range* [""] *use_output_blocks* [0]

Scales the mesh size and edge spacing1, spacing2, and max space. This automatically selects all visible edges.

Usage: **ic_hex_scale_mesh_size_with_spacings_gui***scale_factor* [*parts*] [*index range*] [*use_output_blocks 0*]

Note:

This function runs only in GUI mode, not in batch mode.

ic_hex_scale_mesh_size_with_spacings *scale edges*

Scales the mesh size and edge spacing1, spacing2, and max space.

Usage: **ic_hex_scale_mesh_size_with_spacings***scale_factoredges_to_scale*

Note:

This function runs in both modes (GUI mode and batch mode).

ic_hex _get_edge_segment_number *args*

Returns the number of edge segments (splitted edge).

Usage: **ic_hex_get_edge_segment_number***edge_vertex1edge_vertex2*

ic_hex_get_hidden_nodes *args*

Returns also the hidden nodes (if any) of an edge. Usage: **ic_hex_get_hidden_nodes***edge_vertex1edge_vertex2*

ic_hex_set_unstruct_face_method *args*

Sets the method of mesh for unstructured 2D blocks.

Usage: **ic_hex_unstruct_face_method***method*

Method can be *uniform_quad*, *gambit_pave*, or *auto*.

ic_hex_ogrid_smooth_transition *args*

Turns the Ogrid_smooth_transition option on or off.

Usage: **ic_hex_ogrid_smooth_transition** [*on/off*]

Returns 0 (off) or 1 (on): set state [ic_hex_ogrid_smooth_transition]

ic_hex_project_to_topo *args*

Turns the Project_to_topo option on or off.

Usage: **ic_hex_project_to_topo** [*on/off*]

Returns 0 (off) or 1 (on): set state [ic_hex_project_to_topo]

ic_hex_get_nearest_existing_point *vertex part* `[""]`

Returns the nearest geometry point to a vertex position.

Usage: **ic_hex_get_nearest_existing_point***vertex_num*

ic_hex_compute_curvature_asf *args*

Computes the bunching on the blocking edges according to the TGLib size function background grid (must be accessible).

Usage: **ic_hex_compute_curvature_asf***-asf_mesh_file*`fname`*-do_proximity-skip_edges_in_sweep_dir-skip_dims*`d1 d2 d3 ...`

e.g.,: **eval ic_hex_compute_curvature_asf***-asf_mesh_file*`./asf_mesh.uns`*-do_proximity-skip_dims*`0 4 6`

Argument Descriptions:

-asf_mesh_file	Full path to an unstructured mesh file that stores the edge bunching based on size functions.

Optional arguments:

-do_proximity	Evaluate also the proximity size functions.
-skip_edges_in_sweep_dir	Do not compute the size function based bunching in sweep direction.
-skip_dims	List of integer Hexa blocking dimensions in which direction of the size function should not be evaluated.

ic_hex_count_blocks *type orfn* `[0]`

Returns the number of blocks.

Usage: **ic_hex_count_blocks***block_type<vorfn>*

block_type can be *free*, *mapped*, or *swept*.

Set *vorfn* to 1 to include blocks in part VORFN in the count.

ic_hex_number_all_free_faces

Returns the number of free faces (in 3D) and the number of free blocks (in 2D).

Usage: **ic_hex_number_all_free_faces**

ic_hex_split_unstruct_block3d_by_edge_loops *args*

Split a free block by boundary edges.

Usage: **ic_hex_split_unstruct_block3d_by_edge_loops***ublock_numsheet_num*

ic_hex_convert_to_struct_from_nodes *args*

Convert a free block free face to mapped from four corner vertices.

Usage: **ic_hex_convert_to_struct_from_nodes***-face free_face -v_end "four_corner_vertices"*

Note:

Use this function if the selected free_face has more than four vertices.

---

ic_hex_get_edge_to_apply *n1 n2*

When working with superblocks (blocks that have been merged), some edges have hidden vertices that are used internally in scripting. Returns edge vertices required to define the edge in scripting commands given the visible vertices as input.

Usage: **ic_hex_get_edge_to_apply***start_edge_vertexend_edge_vertex*

Note:

if there are no hidden vertices, the result will be the same as the input. For example, if an edge has visible corners 10 and 20. With a hidden vertex of 15, **ic_hex_get_edge_to_apply***10 20* would return `10 15`. If there are no hidden vertices in the case above, it would return `10 20`.

Meshing Functions

ic_run_tetra *tetin uncut_dom args*

Runs tetra in batch mode, using the given *tetin* and *uncut_domain* files. The full path names must be given. One or more arguments may be given. These arguments are:

- **tradeoff***type*: the type is either *space* or *time*
- **log***logfile* : the file to write stdout to

ic_write_fieldmesher_params_xml_file *working_dir input_mesh output_mesh tetin_file meshing_controls global_maxsize global_minsize global_growthrate smoothing_level use_fluent hexacore inflation last_layer_aspect n_ortho_layers fix_first_layer enhance_norm_comp enhance_offset_comp gap_factor max_cap_skew max_cell_skew frozen_volumes debug interactive*

Creates an XML file that defines the input parameters for the Python FieldMesher controller module. Returns the XML file name.

ic_run_fieldmesher *inpfile outfile args*

Runs the FieldMesher executable in batch mode, using the given *arglist* which is of the form [list options inputfile outputfile]. The full path names for *inputfile* and *outputfile* must be given.

ic_run_tri2tet *inpfile outfile args*

Runs **tri2tet** in batch mode, using the given *arglist* which is of the form [list *optionsinputfileoutputfile*]. The full path names for *inputfile* and *outputfile* must be given.

ic_run_afmesh *inpfile outfile args*

Runs the advancing front mesher in batch mode, using the given *arglist* which is of the form [list *optionsinputfileoutputfile*]. The full path names for *inputfile* and *outputfile* must be given.

ic_run_tgrid *inpfile outfile args*

Runs the Fluent Meshing mesher in batch mode, using the given *arglist* which is of the form [list *optionsinputfileoutputfile*]. The full path names for *inputfile* and *outputfile* must be given.

ic_run_cutter *tetin uncut_dom cut_dom args*

Runs the cutter, using an existing *tetin* file and *uncut_domain* produced by tetra. One or more arguments may be given. These arguments are:

- **saveall***on* : if *on* is 1 (the default) then all elements will be saved, otherwise selected families
- **savefams***f1f2* ... : the families to save if the **saveall** option is 0

- **which***type* : either **volume** (the default) or **surface**
- **fix_nonman***on* : if *on* is 1 then fix non-manifold vertices, otherwise not
- **autovol***0* | *1* : disables (or enables) automatic assignment of a material to unassigned tetrahedra. (D = 0)
- **log***logfile* : the file to write stdout to

Note:

You cannot run cutter unless you load a tetin file first

ic_run_prism *tetin cut_dom prism_dom args*

Runs the prism mesher, using an existing *tetin* file and domain file *cut_dom* produced by cutter. A new domain file *prism_dom* will be created.

The parameter *tetin* is mandatory. If no geometry file should be used, the first argument has to be empty quotes:

- **ic_run_prism** "" *cut_domargs*

One or more arguments may be given. Arguments, default values [...], and optional values /.../ are listed below:

- **layers** [3]
- **height** [0.2]
- **ratio** [1.0] All families will be used for prism as a default. You can define list of families:
- **family** {fam_name fam_name ...} Or you can specify the list with different height and/or ratio for selected families: In both cases parameters height and ratio will be used as defaults.
- **family** {{fam_name height 0.35} {fam_name height 0.3 ratio 1.1} ...}
- **n_triangle_smoothing_steps** [5]
- **n_tetra_smoothing_steps** [10]
- **ortho_weight** [0.5]
- **total_height** []
- **max_aspect_ratio** []
- **prism_height_limit** []
- **triangle_quality** [inscribed_ratio] /laplace height_over_base skewness min_angle max_angle inscribed_area/
- **max_prism_angle** [180]

- **fillet** [0.1]
- **law** [exponential] /linear/wb-exponential
- **into_orphan** 0
- **new_volume** []
- **side_family** []
- **top_family** []
- **smoothing_steps** [6]
- **min_prism_quality** [0.01]
- **max_binary_tetin** [0]
- **use_prism_v10** [0]

Example usage: `ic_run_prism $tetin $domain $new_domain layers 5 height 0.33 family {TOP {WALLS height 0.44} BOTTOM}`

Note:

- Specifying "**use_prism_v10** 1" starts the prism 10.0 version instead of the current version.
- The parameter *tetin* is mandatory. If no geometry file is used, the first argument has to be empty quotes.
  - **ic_run_prism** "" *cut_domargs*

ic_run_hexa *tetin blocking replay proj_dir* [""] *domains_dir* [""] *mesh_dir* [""]

Runs the hexa mesher in batch mode, using the given *tetin* file and *blocking*, which may be "" to start with no blocking, and *replay* file which will be used as the standard input. The project directory should be given as *proj_dir*.

ic_run_frontal *args*

Runs frontal in batch mode, using the given *tetin* file. The full path name must be given. One or more arguments may be given. These arguments are:

- **dir***xyz*: the *xyz* argument is a list of 3 numbers that determines what direction we are looking
- **tol***tol*: the tolerance to use
- **curve***curvefile* : the curve file to write

ic_application_terminate

Kills the current running application.

ic_quad *args*

Free surface meshing by the extended splitting line algorithm. The arguments come in keyword value pairs. Possible options are:

- **name***value*: Name of the mapping
- **q***1/2/3/4/5*:
  - 1 = loop(s) from loop entities (default)
  - 11 = loop(s) from surface entities
  - 2 = loop from bar elements
  - 3 = loop from subset of bar elements
  - 4 = single loop from surface elements
  - 5 = single loop from surface elements, replace old elements if new elements have higher quality, work only on free surface nodes
  - -1 = loop(s) from loop entities, create bar elements only
- **h***1/2/3/4*: Ignore holes if 1, mesh only holes if 2, mesh hull of surface elements if 3, or mesh gaps of surface elements if 4. The default is 0.
- **element***1/2/3/4/5*: If set to 4, every loop must have an even number of nodes on the edges in order to form an all-quadrilateral QUAD_4 mesh. This constraint will be automatically satisfied by creating an even number of nodes on each curve, or it will be dropped if the creation of one (option 1) or several (option 2) TRI_3 transition triangles is accepted. The meshing scheme can also generate TRI_3 triangle surface meshes (option 3), TRI_3 STL-like facets (option 5), structured QUAD_4 grids (option 6), and TRI_3 split STL facets (option 7). The default is 4.
- **inner***value*: Preferred interior element size if > 0. Take from surface parameters if -1. The default is 0.
- **offset***n*: Create n layers of offset elements. The default is 0. Argument "curve_family" can be used to limit offsetting to this family. A negative value will be only applied to curves with a given height parameter.
- **offset_h***value*: Relative value for the offset height. The default is 1. Use -1 to apply the curve parameters.
- **dev***value*: Refine mesh up to three times to reduce surface deviation. Value defines the maximum surface deviation if not 0. The default is 0.
- **proj***0/1*: Project the new nodes onto the CAD surfaces if 1, or onto an isoparametric quadratic interpolation surface if 2. The default is 0.
- **iters***maximum*: The mesh quality can be improved if additional smoothing steps are performed. With this option set, a pure Laplacian smoothing is used in which each interior node is moved to the centroid of its neighbors. In typical cases smoothing will converge after about six steps. The default maximum number of steps is 20. No complaints about inaccuracy if negative.

- **conver***value*: The smoothing accuracy defines a convergence value which is related to the element sizes. The default value is 0.025. Uses an activity control for better performance if negative.
- **b_smooth***0/1* : Smooth boundaries if 1. The default is 0.
- **improvement***0/1/2/3* : Allow advanced improvement operations (node and element elimination, diagonal swapping, element merging and splitting, etc.) if set. The default is 0.
- **violate***value* : Allow a node to move off the surface by this value. The default is 0.
- **merge_tol***value*: Merge tolerance related to element size. The default is 0.1.
- **ele_tol***value* : The default element tolerance is 0. Set this violate geometry value to remove tiny elements, holes and loops. Given bunching will be protected if < 0.
- **boundary_tol***value*: The default boundary tolerance is -1. The mesher then applies above ele_tol on the boundary. Set this value in case you want to have an independent violate geometry tolerance for the boundary elements.
- **geo_tol***value*: The default looping tolerance is 0, which means that curves are only connected via identical endpoints. Set this value to enable free surface meshing to calculate the loop topology of inexact curves.
- **bunch***0/1*: Use node bunching from BAR_2 elements on curves if 1. The default is 0.
- **curve_family***name*: Use curves from this family to define a virtual loop.
- **curve_family_2***name*: Use also curves from this family to define a virtual loop but never associate BAR_2 elements to these curves.
- **error_family***name* : Move failed loops to this family.
- **offset_element_family***name*: Move offset elements to this family.
- **material***name*: Label new elements with this family name.
- **time_max***limit*: Stop the mesher if time limit has been exceeded. Default is infinite seconds.
- **ele_max***limit*: Stop the mesher if the element limit has been exceeded. Default is infinite number of elements.
- **try_twice***0/1/-n*: Try simple triangulation if 1 and meshing fails. First try simple triangulation and then retry without holes if 2. The default is 1. A negative value n limits simple triangulation to loops with number of nodes <= n. (for element type 2 and 3 only)
- **loops***names*: List of selected loops. The default is to mesh all loops.
- **block***value*: Create a structured mesh instead of an unstructured up to this block quality if value > 0. The default value is 0. A suitable choice for hybrid meshes would be 0.2.
- **adjust_nodes***value*: Recompute number of nodes on edges such that all structured loops can be map meshed if set. Smooth nodes if > 1. The default value is 0.
- **adjust_nodes_max***value*: Relative limit for node adjustment. The default value is 0.5 so that only those edges are adjusted where a maximum of 50% of the nodes need to be removed.

- **ignore_tp***0/1*: Ignore location of topological curve end points if set. The default value is 0.
- **board***0/1* : Run improved meshing scheme for boards with many holes if set. The default value is 0.
- **complete_edges***0/1/2*: Complete missing edges in boundary loop if 1. 2 allows more than one loop from bar elements (q = 2 or 3). Broken loops are ignored. The default value is 0.
- **geo_id***0/1*: Create elements with family name of nearest surface if set. The default value is 0.
- **enn***0/1*: Estimated number of nodes if set. The default value is 0.
- **dormant***0/1*: Merge loops at dormant curves if set. The default value is 0.
- **pattern***value*: Improve element pattern up to this angle if > 0. The default value is 150 degree.
- **simple_offset***0/1*: Do simple normal offsetting if set. The default value is 0.
- **four***0/1*: Force four-quad patterns at prescribed points if set. The default value is 0.
- **max_area***value*: Maximum element area if set. The default is 0.
- **max_length***value* : Maximum edge length if set. The default is 0.
- **min_angle***value*: Minimum angle if set. The default is 0.
- **max_nodes***value*: Maximum number of nodes if set. The default is 0.
- **max_elements***value*: Maximum number of elements if set. The default is 0.
- **n_threads***value*: Number of threads. The default is 0.
- **freeb***0/1*: Allow free bunching for patch independent surfaces if set. The default is 0.
- **snorm***0/1*: Orient elements to surface normals if set. The default is 0.
- **shape***0/1*: Element shape function, 0 is linear and 1 is quadratic. The default is 0.

ic_quad_from_geometry *what entities element* [4] *proj* [0] *iters* [20] *conver* [0.025] *tol* [0.1] *ele_tol* [0] *dev* [0] *improvement* [0] *block* [0] *bunch* [1] *violate* [0] *debug* [0] *adjust_nodes* [0] *adjust_nodes_max* [0.5] *max_point_dist* [10] *try_harder* [0] *even_if_dormant* [1] *offset* [-1] *error_subset* [Failed_surfaces] *pattern* [150] *big* [1] *board* [0] *remove_old* [-1] *inner* [0] *simple_offset* [0] *split_tri* [0]

Free surface meshing from geometry. This option creates and deletes its own loops used in the Quad mesher. Provided for backward compatibility. Use **ic_quad2** instead.

- **what**: Can be "surfaces" (creates a loop per surface), "native_surfaces (directly meshes each surface), "connected_surfaces" (creates a loop per connected surfaces), "surface_group" (list of surfaces, where the list of surfaces becomes one loop), "curves" (list of curves becomes a loop), or "curves_only" (list of curves to be meshed with bar elements).
- **entities**: List as described in "what".
- **element**: See **ic_quad** description for **element**.

- **proj** : See **ic_quad** description for **proj**.
- **iters**: See **ic_quad** description for **iters**.
- **conver**: See **ic_quad** description for **conver**.
- **b_smooth**: See **ic_quad** description for **b_smooth**.
- **tol**: See **ic_quad** description for **geo_tol**.
- **ele_tol**: See **ic_quad** description for **ele_tol**.
- **dev**: See **ic_quad** description for **dev**.
- **improvement**: See **ic_quad** description for **improvement**.
- **block**: See **ic_quad** description for **block**.
- **bunch**: See **ic_quad** description for **bunch**.
- **violate**: See **ic_quad** description for **violate**.
- **adjust_nodes**: See **ic_quad** description for **adjust_nodes**.
- **adjust_nodes_max**: See **ic_quad** description for **adjust_nodes_max**.
- **max_point_dist**: For backward compatibility.
- **try_harder***0/3*: Try harder if > 0 and meshing fails (0 = just run the Quad mesher, 1 = try simple triangulation (for element type 2 and 3 only), 2 = try 1 and retry without merging at dormant curves, 3 = try 1 and/or Tetra mesher, and then retry 2). The default is 0.
- **even_if_dormant***0/1*: Use curves even if they are dormant. The default is 1.
- **offset***n*: Create offset elements on curves with a given height parameter if -1. This is the default.
- **pattern***value*: Improve element pattern up to this angle if > 0. The default value is 150 degree.
- **big***0/1*: Undo/redo will be available for each meshing step if 0. Just one undo/redo is stored for 1. The default value is 1.
- **board***0/1* : Run improved meshing scheme for boards with many holes if set. The default value is 0.
- **remove_old***-1/0/1*: Remove old elements if 1, do not remove any elements if 0, use an automatic approach if -1. The default value is -1.
- **inner**: See **ic_quad** description for **inner**.
- **simple_offset**: See **ic_quad** description for **simple_offset**.

ic_quad2 *args*

Free surface meshing from geometry.

ic_quad_debug *db* [0]

Set Quad debug level.

ic_quad_from_elements *name element* [4] *proj* [0] *remesh* [0] *iters* [20] *conver* [0.025] *tol* [0.1] *ele_tol* [0] *dev* [0] *db* [0] *make_consistent* [0] *offset* [0] *offset_h* [1] *keep_fixed_nodes* [0] *inner* [0] *material* [""] *geo_id* [0] *simple_offset* [0]

Free surface meshing from elements.

- **name**: Name of the mapping.
- **element**: If set to 4, every loop must have an even number of nodes on the edges in order to form an all-quadrilateral QUAD_4 mesh. This constraint can be dropped if the creation of one (option 1) or several (option 2) TRI_3 transition triangles is accepted. The meshing scheme can also generate TRI_3 triangle surface meshes (option 3) and TRI_3 STL-like facets (option 5). The default is 4.
- **proj**: Project the new nodes onto the CAD surfaces if 1. The default is 0.
- **remesh**: Ignore holes if 1, mesh only holes if 2, mesh hull of surface elements if 3, or mesh gaps of surface elements if 4. The default is 0.
- **iters**: The mesh quality can be improved if additional smoothing steps are performed. With this option set, a pure Laplacian smoothing is used in which each interior node is moved to the centroid of its neighbors. In typical cases smoothing will converge after about six steps. The default maximum number of steps is 20.
- **conver**: The smoothing accuracy defines a convergence value which is related to the element sizes. The default value is 0.025.
- **tol**: Merge tolerance related to element size. The default is 0.1.
- **ele_tol**: The default element tolerance is 0. Set this violate geometry value to remove tiny elements or holes.
- **dev**: Refine mesh up to three times to reduce surface deviation. Value defines the maximum surface deviation if not 0. The default is 0.
- **db**: Do some checking and printing if > 0, print less messages if < 0. Default is 0.
- **make_consistent** : Make volume mesh consistent if set. The default value is 0.
- **offset**: Create n layers of offset elements. The default is 0.
- **offset_h**: Relative value for the offset height. The default is 1.
- **keep_fixed_nodes**: Keep fixed nodes if set. The default is 0.
- **inner**: Preferred interior element size if > 0. The default is 0.
- **material***name*: Label new elements with this family name.
- **geo_id***0/1*: Create elements with family name of nearest surface if set. The default value is 0.

- **simple_offset***0/1*: Do simple normal offsetting if set. The default value is 0.

ic_quad_from_bars *name element* [4] *proj* [0] *iters* [20] *conver* [0.025] *improvement* [0] *surf* [0] *complete* [0] *ele_tol* [0] *dev* [0] *db* [0] *make_consistent* [0]

Quad from bars.

ic_quad_mesh_surface_from_edge_mesh *sname*

Given a surface name *sname* create a quad mesh from the existing edge mesh on its boundary. Example usage: `ic_uns_quad_mesh_surface_from_edge_mesh E43457`

ic_quad_from_loops *name element proj iters conver improvement bunch loopnames tol etol fam dev db violate* [0] *offset* [0] *offset_h* [1] *offset_element_family* [""] *block* [0] *inner* [0] *boundary_tol* [-1] *adjust_nodes* [0] *board* [0] *ignore_tp* [0] *adjust_nodes_max* [1] *enn* [0] *dormant* [0] *pattern* [0] *b_smooth* [0]

Quad from loops.

ic_smooth_quad *name proj* [0] *iters* [20] *conver* [0.025] *improvement* [0] *type* [4] *w* [0] *b_smooth* [0] *debug* [0] *n_threads* [0] *unique* [0] *diagnostic* [""] *max_aspect_ratio* [-1] *w_transition* [0] *w_first* [0] *h_first* [0] *untangle* [1]

Smooth surface elements. By default, any free surface node is moved to the centroid of its neighbors (Laplacian method).

ic_smooth_hexa *name proj* [0] *iters* [20] *conver* [0.025] *w* [0] *debug* [0] *n_threads* [0] *unique* [0] *diagnostic* [""]

Smooth volume elements. By default, any free surface node is moved to the centroid of its neighbors (Laplacian method).

ic_remesh_edges *name tol merge_ends mtol* [0] *db* [0]

Remesh edges.

ic_remesh_bad_elements *name element* [3] *proj* [0] *db* [0] *all_dim* [0] *max_steps* [-1] *quality* [1] *angle* [0] *ele_tol* [0] *inner* [0] *diagnostic* [Custom quality] *material* [""] *aggressive* [0]

Remesh bad elements.

ic_quad_remesh_bad_elements_by_quality *fams types metric quality_limit* [0.6] *eltype* [2] *ignore_dim* [0]

Remesh regions of a surface mesh by a diagnostic. It will return 1 if it performed remeshing, 0 if it was already okay.

ic_quad_remesh_by_diagnostic *diagnostic fams types nlayers eltype* [2] *ignore_dim* [0] *custom_metric_weights* [""]

Remesh regions of a surface mesh by a diagnostic. It will return 1 if it performed remeshing, 0 if it was already okay.

ic_enforce_vertex *name element* [3] *proj* [0] *db* [0] *all_dim* [0] *vertex* [-1] *make_consistent* [0]

Enforce vertex in surface mesh. The map should include just one node element to provide the vertex to be enforced, or the vertex number must be >= 0.

ic_change_tri_quad_2 *name proj* [0] *db* [0] *all_dim* [0] *max_steps* [-1] *skew* [160] *warp* [20] *quadrization* [0] *diagnostic* [Angle error]

Change tri to quad elements (new scheme).

ic_quad_offset_curve_layers *how which n_layers* [""]

Offset quad layers on the loaded surface mesh. *how* can be one of { curve_family, curve, map } while *which* would be a curve_family, curve_names (list), or map_name respectively. If *n_layers* is not specified, it will try getting the value from the tetin parameters.

ic_mesh_hexcart *args*

Cartesian hexa mesher.

ic_hexcart_shrinkwrap *tetin_file minsize proj_fact* [0.3] *nsmooth* [5] *fam* [inherited] *force_load* [0] *mtype* [0] *batch* [1]

Run Cartesian shrinkwrap.

ic_hexcart_body_fitted *tetin_file minsize parts args*

Create body-fitted Cartesian mesh. If there are more than one part then run part by part.

ic_hexcart_bfcart_inflation

Run BFCart based post inflation for all hex mesh.

ic_mesh_hexdom *family remesh* [0]

Run Hex-Dominant mesher.

ic_run_mapped_based_mesher *type* [Quad Dominant] *surfs* [""] *min_edge* [0] *mode* [""] *no_merge* [1] *version* [100]

Runs mapped based mesher type is type of mesh desired options are "All Quad", "Quad w/one Tri", "Quad Dominant", "All Tri", "Tri (STL like)" Default is "Quad Dominant" surfs is surfaces to mesh, default is all surfaces min_edge is minimum edge length allowed in mesh; default is 0.

Meshing Directives

ic_meshdirect_available_directives_get

Get the list of internal directive names.

ic_meshdirect_get_icon_direction *directive source_family target_family source_type* [""] *target_type* [""]

ic_meshdirect_icons_visibility *family vis*

ic_meshdirect_add_icon *directive source_family target_family* [""] *source_type* [""] *target_type* [""]

ic_meshdirect_delete_icon *directive families*

Removes the icons for a particular family/bctype combination.

ic_meshdirect_delete_icon_on_ents *type ents*

Remove the icons for entities if deleted.

ic_meshdirect_arbitrary_connector_set *source_fams source_type target_fam target_type result_fam extra_args* [""]

Higher-level interface for setting meshing directives for arbitrary connectors.

ic_meshdirect_boltspider_set *bolthole_fams result_fam extra_args* [""]

Higher-level interface for setting meshing directives for bolt-holes/bolt-spiders.

ic_meshdirect_boltspider_set_mesh_sizes *bolthole_fams*

ic_meshdirect_seamweld_set *source_fams target_fam result_fam extra_args* [""]

Higher-level interface for setting meshing directives for seam welding.

ic_meshdirect_spotweld_set *source_fams target_fam result_fam extra_args* [""]

Higher-level interface for setting meshing directives for spot welding.

ic_meshdirect_spotweld_types_verbose_list

Gets the verbose list of current possible types of spotwelds This is good for getting a list of methods for a GUI, each element of which **ic_meshdirect_* procs** will recognize later.

ic_meshdirect_spotweld_types_verbose_get *param*

Returns the *meshing_directive* val for the given verbose human text which should be an element of the list given by **ic_meshdirect_spotweld_types_verbose_list**.

ic_meshdirect_seamweld_types_verbose_list

Gets the verbose list of current possible types of spotwelds This is good for getting a list of methods for a GUI, each element of which **ic_meshdirect_* procs** will recognize later.

ic_meshdirect_seamweld_types_verbose_get *param*

Returns the *meshing_directive* val for the given verbose human text which should be an element of the list given by **ic_meshdirect_seamweldtypes_types_verbose_list**.

ic_meshdirect_spotweld_types_param_get *verbose_type*

Returns the *meshing_directive* tag for the given verbose human text which should be an element of the list given by **ic_meshdirect_spotweld_types_verbose_list**.

ic_meshdirect_seamweld_types_param_get *verbose_type*

Returns the *meshing_directive* tag for the given verbose human text which should be an element of the list given by **ic_meshdirect_seamweld_types_verbose_list**.

ic_meshdirect_weld_splitting_verbose_list

Gets the verbose list of current possible splitting options for spotwelds This is good for getting a list of methods for a GUI, each element of which **ic_meshdirect_* procs** will recognize later.

ic_meshdirect_weld_splitting_param_get *verbose_type*

Returns the *meshing_directive* tag for the given verbose human text which should be an element of the list given by **ic_meshdirect_spotweld_types_verbose_list**.

ic_meshdirect_weld_splitting_verbose_get *param*

ic_meshdirect_glinka_side_param_get *verbose_type*

ic_meshdirect_glinka_subtype_verbose_get *param*

Gets verbose glinka subtypes.

ic_meshdirect_glinka_side_verbose_get *param*

ic_meshdirect_directive_set_vals_by_tag *directive fams vals*

This is for use by MED GUI.

ic_meshdirect_directive_set_val_by_tag *directive fams tag val*

This is for use by MED GUI.

ic_meshdirect_directive_get_val_by_tag *directive fam tag*

This is for use by MED GUI.

ic_meshdirect_model_has_directives

Determines if the loaded tetin has meshing directives defined.

ic_meshdirect_get_directive_of_fam *famnames*

Returns the meshing directive defined on a family, or the first found for a list of families.

ic_meshdirect_directive_uses_family *given_fam*

Returns whether or not the given family is used in any meshing directives.

ic_meshdirect_param_requires_remeshing

Returns whether or not the given family is used in any meshing directives.

ic_meshdirect_directive_tags_get *directive*

This is for use by MED GUI.

ic_meshdirect_spotweld_file_import *filename args* [""]

----> Weld importing needs doc.

ic_meshdirect_standalone_exec *idomain tetin odomain* [""] *args*

Executes meshing directives. Run meshing directives, using the given input mesh *idomain* and *tetin* files and, optionally, the output mesh *odomain*. The full path names must be given. One or more directive arguments may be given. These arguments are:

- *-spotweld* : run the spotweld directives defined in the tetin
- *-seamweld*: run the seamweld directives defined in the tetin
- *-boltspider*: run the boltspider directives defined in the tetin
- *-connector*: run the connector directives defined in the tetin
- *-pml*: run the pml directives defined in the tetin

ic_meshdirect_object_exec *types args* [""]

Run meshing directives, using the mesh and proj objects in memory. Any number of these directive types may be given:

- *spotweld*: run the spotweld directives defined in the tetin
- *seamweld*: run the seamweld directives defined in the tetin
- *boltspider*: run the boltspider directives defined in the tetin
- *connector*: run the connector directives defined in the tetin
- *pml*: run the pml directives defined in the tetin

Additionally, options can be specified in *args* such as:

- *-remesh_affected_elements*: remesh elements affected by directive

ic_meshdirect_weld_prepoint_standard *ppname args*

Creates a standard (BAR/LINE) weld from the prescribed point *ppname* in source family *source_fam* to the target family *target_fam*, putting the weld in *weld_fam*.

ic_meshdirect_weld_prepoint *ppname args*

Creates a weld of some type from the prescribed point *ppname* in source family *source_fam* to the target family *target_fam*, putting the weld in *weld_fam*.

ic_meshdirect_spot_weld *args*

Creates a spot weld using the following flags as arguments:

One of these methods for source point location: @ -location {%F %F %F}

Coords of location to weld at @ -prepoint %S

Name of prepoint of location to weld at, kind of weld to create there: @ -method %S one of {*standard*, *meshless_hex*, *meshless_area*, *meshless_bar*, *shadow*}, (default is *standard*)

A bit more on how to create it: @ -maxprojection %F maximum length of weld (default is infinity) @ -element_splitting %S one of { *terminate*, *propagate*, *remesh_all_quad*, *remesh_tri_quad*}; (default is *terminate*)

For an AreaWeld, you can specify how large an area: @ -radius %F radius of area_weld coverage.

To manage the parts/families which it connects: @ -target_family %S family to which to weld @ -source_family %S family from which to weld (syn. of -target_family). Note that target can have multiple definitions, each of which will be targeted.

To specify what to do with new elements: @ -spotweld_family %S family for weld elements to go into @ -rbe3_family %S family for rbe3 element to go into

ic_meshdirect_spot_weld_shadow_points_create *args*

Creates shadow points for the given arguments.

ic_meshdirect_entity_is_shadow_entity *ent_name*

ic_meshdirect_spot_weld_shadow_points_on_surface *srf*

Gets the shadow welds that have been created on the given surface *srf*.

ic_meshdirect_seam_weld_shadow_curves_create *args*

Creates shadow curves in the geometry for the seam weld described in the arguments. Example *args*: "-source_family $locFamOnWhich -target_families $seamTargetFam"

ic_meshdirect_seam_weld_shadow_curves_on_surface *srf*

Gets all of the shadow curves which have been created on the given surface *srf*.

ic_meshdirect_enforce_curve_shadow_nodes_on_surfaces *source_curves target_surfaces*

ic_geo_crv_parent_srf_is_coplanar_to_nearest *crv srfs*

Returns whether or not the surface incident to *crv* is coplanar to all of the surfaces in *srfs*.

ic_meshdirect_seam_weld_glinka_geo_create *source_curves target_family extra_args*

Returns the list of curves on which the actual directive should be defined. Those curves are not necessarily the *source_curves* since it is not known whether you will be doing a *surface_extend* (in which case return the same *source_curves*) or a *curtain_surface* (in which case return different *source_curves*).

ic_meshdirect_seam_weld_glinka_geo_valid *glinka_source_family*

ic_meshdirect_ensure_good_curtain_connectivity *weld_families*

ic_meshdirect_setlist *directive list*

ic_meshdirect_set *directive flexentry*

ic_meshdirect_reset *directives* [""] *fams* [""]

Resets all of the *mesh_direct* parameters in the tetin file.

ic_meshdirect_get_current_for_flexlist *directives* [""] *fams* [""]

Returns the list of parameters from the tetin file, in the style of the flexlist.

ic_meshdirect_find_curve_contacts *distance parts* [""] *weld_args* [""]

This procedure finds curves in a part within a distance of other parts. *distance* is distance between parts. *parts* is a list of parts to check. *weld_args* are the weld arguments that need to be passed to **ic_meshdirect_seamweld_set**

Structured Mesh Editing and Modification Functions

ic_str_is_loaded

Determines if a mesh exists.

ic_str_is_modified

Determines if the mesh has been modified.

ic_str_set_modified *mod*

Set the structured mesh modified flag.

ic_str_families_changed

Indicates something has changed with the families - color, etc. Update all the colors on all the surfaces.

ic_str_ijk_limits *num*

Returns the ijk limits of a domain.

ic_str_domain_dimension *num*

Returns the dimension of this domain.

ic_str_domainfile_dimension *domain*

Returns the dimension of the domain file without loading the domain.

ic_str_get_topo_info *what num full*

Returns the topology info for this object.

ic_str_get_family *what num*

Gets the family for an object.

ic_str_get_name *what num*

Gets the name of an object.

ic_str_get_ijk *what num*

Gets the IJK ranges of an object.

ic_str_list_in_families *what fams*

Lists the objects in some families.

ic_str_count_in_families *fams*

Returns the number of objects are in a given family.

ic_str_adjacent_surfs *what num*

Lists the adjacent objects for this object - currently only adjacent domains to this domain are supported.

ic_str_boundaries_of_entity *what num*

List the boundary subfaces for this domain.

ic_str_load *domain_prefix topofile famtopofile subfaceprojfile suffices* [""] *auto_topo* [0]

Load the structured mesh in the given *dir*. The argument *auto_topo* will force a topology to be created automatically if it does not exist.

ic_str_readexternal *backend nfiles files create* [1] *prec* [1] *tstep* [0] *args*

Import a number (*nfiles*) of structured mesh files (*files*) into a mesh object using the specified **backend**. The optional argument *create* (*default: 1*) allows one to overwrite the current mesh object, while *prec* (*default: 1 [single]*) allows one to specify the real-number precision. One can also specify the time step *tstep* (*default: 0*) and any additional arguments (*args*) necessary for the import.

ic_str_settopo *merging ask* [1]

Sets the topology.

ic_str_summary

Print a summary of the domains, etc, to the message window.

ic_str_save *domain_dir topofile famtopofile subfaceprojfile dompref* [domain.]

Saves a structured mesh.

ic_save_domain_proj *proj_dir*

Save a structured mesh with default filenames in the given project directory.

ic_str_num_entities *type*

Returns the number of objects of a given type.

ic_str_list_entities *type*

List the numbers of the objects of a given type.

ic_str_list_entities_visible *type vis*

List the numbers of the objects of a given type only if visible or blanked.

ic_str_list_families

Lists the families in this mesh.

ic_str_clear_diagnostics

Clear all the diagnostics from a structured mesh.

ic_str_metric *type domains*

Compute diagnostics.

ic_str_create_diagnostic_subset *vis* [1]

Creates a diagnostic cell set.

ic_str_configure_diagnostic_subset *args*

Configures the diagnostic cell set.

ic_str_set_with_diagnostic *intervals domains*

Configures the diagnostic cell set.

ic_str_histogram *min max nbars domains*

Use the diagnostic data for a histogram.

ic_str_copy_domain *num*

Make a new domain.

ic_str_move_domain *num args*

Move a domain. The *num* argument gives the domain to operate on.

- **cent** : a list of X Y Z giving the center for rotation, scaling, and mirroring, which could be "centroid"
- **translate** : the X Y Z values for translating the entity
- **rotate** : the number of degrees to rotate the object about the axis given by **rotate_axis**
- **rotate_axis** : the vector giving the axis to rotate about
- **scale** : a scale value, or 3 values giving the factor to scale in X Y and Z
- **mirror** : the axis about which to mirror
- **flip_ijk** : a triple of 1's and 0's that says whether to flip the IJK axis

- **permute_ijk** : a permutation of 0 1 2

ic_str_create_selection_subset *vis* [1] *nodes* [0]

Creates a selection cell set, or a node selection surface.

ic_str_clear_selection_subset

Clears the selection cell set.

ic_str_selection_numnodes

Returns the number of nodes.

ic_str_selection_get_bbox

Returns the bounding box of the selection.

ic_str_domain_orient *num*

Checks the orientation of a domain.

ic_str_align_all_ijks *num stop_num*

Align all domains with one domain.

ic_str_update_surfs

Update all the graphical images.

ic_str_domain_centroid *num*

Return the centroid of a domain.

ic_str_merge_subfaces *num1 num2*

Merges two subfaces.

ic_str_split_subface *num*

Splits one subface.

ic_str_change_family *type nums fam*

Change the family for an entity.

ic_str_topo_make_surfs *stype force*

Creates the surfaces of a particular type.

ic_str_recompute_topo *do_assign* [1] *symfams* [""]

Recomputes the topology.

ic_str_assign_subface_families

This step has to happen before copying domains.

ic_str_dump_topo_info *file*

Dumps the topology information to a file.

ic_str_configure *what nums shade type_color nodes ijk names numbers outlines node_size famcols simplify ijk_shrink width extended changes draw_twin*

Change the properties of some surfaces.

ic_str_highlight *what nums on color* [""]

Temporarily highlight one surface.

ic_str_set_nums_visible *type nvals*

Enable/disable one surface.

ic_str_set_type_family_visible *tvals fvals*

Enable/disable multiple families If *fvals* is blank then enable/disable all types as specified, likewise for *tvals*. Note that *fvals* == "" means enable everything even if it is not in any family.

ic_str_scanp_create *name color*

Create a new scan plane.

ic_str_scanp_set *name dom ijk ind ext color*

Set a scan plane.

ic_str_scanp_move *name dir ext*

Move a scan plane.

ic_str_scanp_configure *name args*

Get or set the config params on a scan plane. If *name* is "" then this applies to all scan planes.

ic_str_scanp_delete *name*

Delete a scan plane. If name is "" then delete them all.

ic_str_scanp_list

List the active scan planes.

ic_str_scanp_exists *name*

Check whether a scan plane exists.

ic_str_pick_polygon *type poly how mode*

Pick structured items based on a polygon, *how* can be item or node.

ic_str_pick_single *type x y how*

Pick structured items based on a screen position. *how* can be item or node.

ic_str_add_pick_polygon *what poly vect partial*

Pick structured nodes based on a polygon, and add them to the selected node surface.

ic_str_add_pick_single *what pt vec*

Pick structured nodes based on a single location, and add them to the selected node surface.

ic_str_pick_scan_plane_nodes

Add all nodes from the visible scan planes to the selection.

ic_str_uniqify_selection

Uniquify the selection.

ic_str_push_node_positions

Push node positions on the selected surface.

ic_str_clear_node_positions

Clear node positions on the selected surface.

ic_str_pop_node_positions

Pop node positions on the selected surface.

ic_str_drag_nodes *startpt pt vec do_proj allow_invert*

Drag nodes.

ic_str_pick_remove_last

Remove the last selection.

ic_str_check_blockif_surf *type number*

Check for blockinterface.

ic_str_list_blockif_subfaces

Return all blockinterfaces subfaces.

ic_str_list_blockif_surfs *type*

Return all blockinterfaces of type.

ic_str_list_boundary_subfaces

Return all boundary subfaces.

ic_str_glb_sm_init

Done smoothing (glsm).

ic_str_post_diagnostic *type list show*

Post smoothing - create a cellset (glsm).

ic_str_post_add_diagnostic *iv list*

Post smoothing - add a cellset (glsm).

ic_str_create_zone *color shade*

Create a non-relaxation/relaxation zone (glsm).

ic_str_delete_zone *im*

Delete the ijk zone (glsm).

ic_str_configure_zone *im param value*

Configure the ijk zone (glsm).

ic_str_clear_ijk_zone *im*

Clear the ijk zone (glsm).

ic_str_add_ijk_zone *im dom min max*

Add an ijk zone (glsm).

ic_str_scanp_move_zone *type number what ext*

Move an ijk zone (glsm).

ic_str_get_surf_zone *type number*

Get image of zone (glsm).

ic_str_smooth_glsm *change project smooth_domain face_methods volume_methods niter nsteps maxlevel sfit conditional check_det damping vertex_fix edge_freeze wedge_check multigrid2d reduced_edge fix_fam inner_vol_sm ns_rel angle_freeze_edge n_iter interp_faces interp_volumes prog_select threshold_det threshold_angle subface_extensions ijk_extensions edge_attrib relax_subfaces relaxation_zones viol_geom_zones viol_geom_subfaces base_subfaces_no meshing_law subface_extend direction_vector base_regions type_improve type_postsm proj_limit group_bad_elem smooth_dir mesh_type smooth_ngh_method surfresolv min_mxr edge_no_global_smooth incr_3d_plane_smooth relax_weights_2d relax_weights_3d relax_weights_inner_2d relax_weights_inner_3d n_iter_glb n_steps_glb global_vertices_opp plane_numbers grid_exp_face grid_exp_vol min_res abs_dist_bnd n_points_abs_dist_bnd abs_dist_subf n_abs_dist_subf vertex_attrib relaxation_factor use_orthog_position use_fract_position grid_exp_rate_subf_perp grid_exp_rate_subf_1 first_cell_height last_cell_height use_pthreads num_procs* [0] *hostfile* [""] *timeout* [0] *verbose* [0]

Struct smoothing / predefined mesh (glsm).

ic_str_smooth_glsm_clear

After struct smoothing/predefined mesh (glsm) clear all.

ic_str_get_num_nodes *what num*

Get number of nodes.

ic_str_get_bbox *what num*

Get bounding box.

ic_str_move_nodes_exact *what num set_pos dox doy doz xv yv zv*

Move nodes.

ic_str_list_vertex_edges *numbers*

Return all edges from a list of vertices.

ic_str_list_edge_vertices *numbers*

Return all vertices from a list of edges.

Unstructured Mesh Editing and Modification Functions

ic_uns_is_loaded

Checks if a mesh exists.

ic_uns_is_modified

Checks if the mesh has been modified.

ic_uns_set_modified *mod*

Sets the mesh modified flag. This should not be used for most operations since they set this flag themselves.

ic_uns_load *files maxdim* [3] *quiet* [0] *reset_family_prefix* [""] *check_orient* [1]

Loads the given unstructured domain *files*. This must be a full pathname. If *maxdim* is given this will be the maximum dimension of elements that will be loaded. For example, a value of 2 means to load only surface and bar elements. If *check_orient* is 0 then no orientation check will be done. If it is 1 then the code will try to pick a good family for the new surface elements. If it is 2 then the name will be CREATED_FACES.

ic_uns_readexternal *backend nfiles files create* [1] *prec* [1] *tstep* [0] *args*

Imports a number (*nfiles*) of unstructured mesh files (*files*) into a mesh object using the specified *backend*. The optional argument *create* (*default: 1*) allows one to overwrite the current mesh object, while *prec* (*default: 1* [single]) allows one to specify the real-number precision. One can also specify the time step *tstep* (*default: 0*) and any additional arguments (*args*) necessary for the import.

ic_uns_create_empty

Creates an empty mesh if there is not already one available. Returns 1 if it created a new one and 0 if there was an existing one.

ic_save_unstruct *file inc_subs* [1] *resnames* [""] *only_types* [""] *only_fams* [""] *only_subs* [""] *near_vols* [0]

Saves the current unstructured mesh to the given *file*. If the *inc_subs* argument is 1 (the default) then the current subsets are also saved. *resnames* is an optional list of result names to save with the domain (assuming they have been defined in the mesh). If *only_subs* is non-empty it is a list of names of maps that will be the only ones saved.

ic_uns_print_info *what name* [""] *opt* [0] *subset_name1* [max_edge_sides] *subset_name2* [min_edge_sides]

Prints some information. The argument *what* is one of **summary, element, node, or domain**. For **summary** and **domain**, the **name** argument is ignored. For **summary** the argument *opt* will be used to print min/max edge sides if set

ic_uns_check_duplicate_numbers *skip_0* [1]

Checks for duplicate element and node numbers. Returns a list with two numbers - the number of dup elements and the number of dup nodes. Note that 0-numbered nodes and elements are not counted if skip_0 is 1 (the default).

ic_uns_lookup_family_pid *fam*

Returns the numeric ID for this family.

ic_uns_rename_family *old new*

Renames a family. Note that this does not need to be called any more if you call **ic_geo_rename_family** first.

ic_uns_count_in_families *fams total* [0]

Counts the number of elements in each family.

ic_uns_non_empty_families *prefix* [""]

If prefix is default "", all the non-empty families will be listed. If a prefix is specified (not ""), only non-empty families with this prefix will be listed.

ic_uns_count_family_neighbors *fams*

Returns a list of the number of 0-sided, 1-sided, and 2-sided elements in the given families.

ic_uns_family_has_quadratic *fam*

Checks if the family has quadratic elements.

ic_uns_subset_add_by_neighbors *fams to nsides*

Adds to a subset by neighbors.

ic_uns_families_changed

Indicates something has changed with the families - color, etc.

ic_uns_check_family *name*

Checks whether a family exists.

ic_uns_move_family_elements *old_fam_name new_fam_name*

Moves elements from one family (*old_fam_name*) to another (*new_fam_name*).

ic_uns_new_family *name vis* [1]

Creates a new family if it is not already there. Return 1 if a new family was created.

ic_uns_delete_family *fams*

Deletes a family, or list of families.

ic_uns_list_types

Returns the types in the current mesh.

ic_uns_list_possible_types

Returns a list of the possible types in the current mesh.

ic_uns_num_verts_in_type *type*

Returns the number of verts in this type.

ic_uns_dimension

Returns the dimension of the mesh.

ic_uns_list_families *with_dim* [""]

Returns the families in the current mesh.

ic_count_elements *type* [MIXED]

Counts the number of elements in the given type. If the type is MIXED, which is the default, then all elements will be counted. The type may be one of: **0d 1d 2d 3d node line tri quad tetra hexa penta pyra** in which case those types of elements will be listed.

ic_count_nodes

Counts the number of nodes in the mesh.

ic_uns_num_couplings

Counts the number of couplings.

ic_uns_project_to_geometry *mapn type*

Returns the names of the surfaces or curves that the elements in the map are closest to. 2D elements get projected to surfaces, 1D to curves. *type* is either surface or curve.

ic_uns_create_element *type fam nodes make_consistent cmapname* [visible] *grab_existing* [0] *reset_dim* [1] *update* [1] *automatic* [1]

Creates a new element, or in the case of NODE elements, as many NODE elements as there are vertices given.

- **type** : the type of the element (NODE, BAR_2, TRI_3, etc)
- **fam** : the family for the new element
- **nodes** : the name of a subset that contains the necessary nodes
- **make_consistent** : if 1, modify the volume mesh to make it consistent with the new element (if it's a surface element)
- **cmapname** : the name of the subset to add the new elements to
- **grab_existing** : if this is 1 and there are already elements like this available then add them to the map
- **reset_dim** : if this is 1, nodes projection will be reset

ic_uns_create_hexa_from_faces *name faces inherit_part* [1] *part* [""]

Creates hexa from opposite faces.

ic_uns_create_node_element_near_point *pname fam*

Creates a point element at a particular node, which is closest to the named prescribed point.

ic_uns_change_family *name fam*

Sets the elements in a subset to a family.

ic_uns_set_part *name newpart*

Moves entities from one part to another.

ic_uns_set_projected_family *mapname*

Sets the family of elements to that of geometry entities they are projected to.

ic_uns_change_family_if_project *mapname ofam nfam geotype geoname*

Changes the elements in a family to a new family, *if* when they are projected to surfaces, curves, etc. they are nearest to a specific one. Returns the number of elements changed.

Example: `ic_uns_change_family_if_project all OWALL NWALL surface NWALL.1`

This works for surfaces, curves, and points.

ic_uns_get_attached_elements *geotype geoname mapname* [""]

Finds the elements attached to a geometry entity *if* when they are projected to surfaces, curves, etc. they are nearest to a specific one. Returns subset of attached elements.

This works for bodies, surfaces, curves, and points.

ic_uns_list_material_numbers

Gets material numbers.

ic_uns_list_material_families

Gets material families.

ic_uns_split_elements *name*

Splits elements.

ic_uns_split_spanning_edges *name args* [""]

Splits spanning elements.

ic_uns_split_edges *name propagate* [0] *project* [1] *check* [0]

Splits edges. If *propagate* is specified, the split edge will propagate through elements, creating no new element types; if propagate==0, then it may, for example, split QUADs into TRIs. *project* can be set to 0 to disable projection. *check*: if 1, then check if split of any neighbor element is not supported

ic_uns_swap_edges *name try_harder*

Swaps edges.

ic_uns_swap_edges_node_numbers *node_numbers try_harder*

Swaps the edges in a given TCL list *node_numbers* arranged as shown: {edge1v1 edge1v2 edge2v1 edge2v2 ...}

ic_uns_swap_edges_auto *name minasp numiter maxdev* [-1.0]

Swaps edges automatically.

ic_uns_change_type *name from to proj cons* [0] *normal* [0]

Changes element types.

ic_uns_change_shell_solid *name thickness adjust hext*

Changes element types shell to solid.

ic_uns_change_type_by_quality *fams from to metric quality_limit* [0.2]

Changes all elements in family *fams* of type *from* to type *to* if below scalar *quality_limit* according to metric *metric*. Only supports QUAD_4 and TRI_3 so far.

ic_smooth_elements *args*

Smooths the current unstructured mesh. One or more arguments may be given. These arguments are:

- **smooth***type* : Smooth all vertices attached to elements of the named type.

- **freeze***type* : Freeze all vertices attached to elements of the named type.
- **float***type* : Allow all vertices attached to elements of the named type to float - that is, they will move only if they are attached to an element type marked **smooth**, but not if they are also attached to an element type marked **freeze**. This is the default state for all types except **NODE** elements which are frozen.
- **family***famname* : smooth elements of this family. This argument can be given more than once for multiple families. If no family is given then all families are selected.
- **metric***val* : The quality metric to optimize. The default is "Determinant".
- **upto***val* : The highest quality element it will try to improve. The default is 0.5.
- **iterations***val* : The number of iterations. The default is 5.
- **prism_warp_weight***val* : The prism warp weight. The default is 0.5.
- **laplace***on* : Smooth using the Laplace algorithm. The default is 0. If set to 2, edge length Laplace is used.
- **no_collapse***on* : Disallow nodes to be merged. The default is 0.
- **sfit***on* : Enable surface fitting when smoothing QUAD_4 and HEXA_8 elements. The default is 0.
- **ignore_pp***on* : Ignore prescribed points when smoothing QUAD_4 and HEXA_8 elements. The default is 0.
- **group_bad_elem***on* : Group bad elements 0/1. The default is 1.
- **only_triangles***on* : Only smooth triangles. The default is 0. **Obsolete option - use float, freeze, or smooth.**
- **smooth_prisms***on* : Smooth prisms. The default is 0. **Obsolete option - use float, freeze, or smooth.**
- **fix_prisms***on* : do not smooth prisms. The default is 1. **Obsolete option - use float, freeze, or smooth.**
- **refine***on* : Use refinement. The default is 0. **Obsolete option - use float, freeze, or smooth.**
- **fix_triangles***on* : do not move triangles. The default is 0. **Obsolete option - use float, freeze, or smooth.**
- **smooth_tets***val* : Smooth tetrahedra. The default is 1. **Obsolete option - use float, freeze, or smooth.**
- **fix_tets***val* : do not smooth tetrahedra. The default is 0. **Obsolete option - use float, freeze, or smooth.**

ic_uns_smooth_by_quality*famstypesmetricquality_limit* [0.5] *additional_args* [""]

Smooths all elements in family *fams* of types *types* if below scalar *quality_limit* according to *metric*.

ic_delete_elements *args*

Deletes selected elements from the loaded unstructured mesh. The arguments come in keyword value pairs. The possible options are:

- **type***typename* : delete elements of this type where typename is TETRA_4 or HEXA_8 or another standard name. If no type name is given then all types are selected.
- **family***famname* : delete elements of this family. If no family is given then all families are selected.

If multiple types are given then all types selected are deleted, and likewise for families.

ic_move_nodes *args*

Moves nodes in the selected elements to the given location. The arguments come in keyword value pairs. The possible options are:

- **type***typename* : delete elements of this type where typename is TETRA_4 or HEXA_8 or another standard name. If no type name is given then all types are selected.
- **family***famname* : delete elements of this family. If no family is given then all families are selected.
- **how***mode* : can be either **set** or **delta**.
- **x***x* : the x value to set or increment.
- **y***y* : the y value to set or increment.
- **z***z* : the z value to set or increment.

If multiple types are given then all types selected are moved, and likewise for families. If any of the X Y or Z values are not given then that coordinate is not changed.

ic_uns_project_nodes_to_line *pt1 pt2 unset_dim name*

Projects all the nodes in a map to the line defined by the two points. This also sets the type to volume so they are no longer associated with any geometry (if unset_dim != 0)

ic_uns_project_nodes_to_plane *pt norm name*

Projects all the nodes in a map to the plane defined by a pt and norm.

ic_change_family_elements *args*

Changes the family of selected elements from the loaded unstructured mesh. The arguments come in keyword value pairs. The possible options are:

- **type***typename* : select elements of this type where typename is a standard element type such as TETRA_4 or HEXA_8. If no type name is given then all types are selected.
- **family***famname* : select elements of this family. If no family is given then all families are selected.
- **newfam***famname* : the family to switch to. This option must be given.

If multiple types are given then all types selected are selected, and likewise for families.

ic_change_linear_quadratic *args*

Changes the selected elements from linear to quadratic.

ic_change_quadratic_linear *name refine keep_interf* [0]

Changes the selected elements from quadratic to linear.

ic_change_tri_quad *args*

Converts all TRI_3 elements into QUAD_4 elements.

ic_extrude *args*

Extrudes selected elements from the loaded unstructured mesh. The arguments come in keyword value pairs. The possible options are:

- **type***typename* : extrude elements of this type, which can be TRI_3 or QUAD_4. If no type name is given then all types are selected.
- **family***famname* : extrude elements of this family. This argument can be given more than once for multiple families. If no family is given then all families are selected.
- **dir***direction* : elements should grow in this direction. It can take any one of the following strings: normal - Normal to the elements, curve_axial - Along curve dir axially, curve_normal - Along curve dir normal to the curve tangent, "X Y Z" - Arbitrary Direction
- **curve***curvename* : extrude along the curve
- **curvedir***[0,1]* : direction = 1 if the elements to be extruded in the reverse direction of the curve.
- **twist***twist* : twist angle per layer.
- **space***spacing* : spacing between adjacent layers.
- **numlayers>***numlayers* : number of layers to extrude.
- **volf***volume_family* : family for the new volume elements.
- **sidef>***side_family* : family for the exposed sides.
- **topf***top_family* : family for the top of the extrusion.
- **del_orig***on* : if on is 1 delete the original surface
- **save_verts***on* : if on is 1 then save temporary data so that another extrusion can be done in the same area.
- **save_node_map***name* : if given this is the name of an UnsMap object that will contain the list of new nodes that are created (and the bottom layer which already exists).

ic_uncouple_main *args*

This is the old "standalone" program of uncouple which can be launched from the classic ICEM CFD Manager ("icemcfd -3").

ic_test_mesh *id* [""] *min* [0] *tol* [0]

Regression testing for unstructured meshes using a "unique" ID string.

ic_test_mesh_quality *min* [0] *tol* [0]

Tests the quality of unstructured meshes.

ic_test_premesh *id* [n_hexa-n_quad-0-0-n_bar-n_coupling-n_vertex-quality] *fam* [""] *tmp_file* [ic_test_premesh_tmp.uns]

Regression testing for premeshes using a "unique" ID string.

ic_test_premesh_quality *min* [0] *tol* [0] *fam* [""] *tmp_file* [ic_test_premesh_tmp.uns]

Tests the quality for premeshes.

ic_uns_length *name* [visible]

Calculates the length of the given mesh. *name*: name of map of the selected elements.

ic_uns_area *name* [visible]

Calculates the area of the given mesh. *name*: name of map of the selected elements.

ic_uns_volume *name* [visible]

Calculates the volume of the given mesh. *name*: name of map of the selected elements.

ic_worst_angle

Find worst angle in surface mesh. Returns maximum corner error or warpage.

ic_merge_meshes *merge_fams fixed_fams remove_interface* [0]

Merges the mesh elements. The two lists are the family names for the surfaces to be merged and the family names for the volume families (if any) to be kept fixed.

ic_struct_to_unstruct *which dims all_elems*

Converts a structured mesh to unstructured.

- **which** is a list of domain numbers - blank if all
- **dims** is the max dimension to convert - 3 means everything
- **all_elems** means to convert internal subfaces, edges, and vertices which are not projected to families into unstructured elements also (which will be in the ORFN family)

ic_uns_move_mesh *origin axis trans theta mm_cmd surfs fsurfs vols*

Moving meshes.

ic_uns_uncouple *which proj* [0] *three* [0] *db* [0]

Uncouples hanging nodes of hexahedra. Users have to take care that the ratio of refinement parameters of adjacent faces/blocks has to be 1, 3, 9, 27, 81 ... The refinement scheme is stable, i.e. the minimum angle does not depend on the subdivision level. Concave regions are filled with a fine mesh. 3-1-1 refinement is not supported. 3-3-1 works for 2.5D cases, 3-3-3 is fine in general.

- **proj**: Do surface projection if not 0.
- **three**: Pure 3-3-3 refinement if > 0; allow unstable patterns if < 0.
- **db**: Do some checking and printing if > 0. Default is 0.

ic_uns_redistribute_prism_edge *height fix_height* [1] *ratio* [0.0] *local* [0] *end_aspect* [0.3] *ignore* [0]

Redistributes prisms.

ic_uns_split_prisms *which layers ratio layer_number* [""] *height* [""] *fams* [""] *fams_srf* [""] *skip_pyras* [""]

Splits prisms.

ic_uns_dissolve_prisms *which height_ratio prism_qual triangle_qual clear_tops*

Dissolves prisms.

ic_uns_convert_to_hexas *min_aspect* [1] *volfams* [""] *use_active_lcs* [0]

Converts tetras to hexas. It will not be done if the hexas have aspect ratio worse than *min_aspect*.

ic_uns_convert_to_hexas_in_batch *infile outfile min_aspect* [1] *volfams* [""] *use_active_lcs* [0]

Converts tetras to hexas in batch mode from input file. It will not be done if the hexas have aspect ratio worse than *min_aspect*.

not yet implemented append argv " \"convert_to_hexas min_aspect $min_aspect $volfams $use_active_lcs\""

ic_uns_coupling *what args*

Creates or deletes couplings

ic_uns_thickness *what name* [all] *val* [0] *strict* [1] *from_solid* [0] *surfs* [""]

Creates or deletes thickness for surface mesh. *from_solid* = 1 = nearest point projection, *from _solid* = 2 = piercing

ic_uns_thickness_at_node *name val*

Applies thickness on specific nodes.

ic_uns_bar_orientation *what name* [all] *x1* [0] *x2* [0] *x3* [0] *wa1* [0] *wa2* [0] *wa3* [0] *wb1* [0] *wb2* [0] *wb3* [0]

Creates or deletes bar orientations for line elements.

ic_uns_list_strings

Lists the strings in the domain file.

ic_uns_set_string *name val*

Sets a string in the domain file.

ic_uns_split_double_wall *which*

Splits double walls.

ic_uns_split_internal_faces *which fam new_fam_suffix* [""] *splitfam* [0] *volfam* [""]

Splits internal faces.

ic_uns_make_consistent *which count_open* [""] *flood_fill* [1]

Makes consistent.

ic_uns_check_edges *which*

Checks edges.

ic_uns_split_pyras *which*

Splits pyramids.

ic_uns_check_orientation *which* [All] *err_name* [""] *do_fix* [0] *quiet* [0]

Checks orientation of the subset *which* which is, by default, **All**. Returns 1 for OK. *err_name* is the name of a subset which will be created if there are errors

ic_uns_orient *dir*

Changes orientation.

ic_uns_reorient *name how args*

Changes orientation.

ic_uns_reorder *nme how args*

Reorders elements.

ic_uns_max_vertex_degree *which*

Returns the max vertex degree.

ic_uns_coarsen *which aspect fixfams tol n_iter surface size_check max_size*

Coarsens the mesh.

ic_uns_coarsen_quad *name* [visible] *dev* [0.1] *edge_len* [-1] *steps* [1]

Coarsens quad dominant mesh

ic_uns_merge_nodes *name tol pos* [0 0 0] *norm* [0 0 0] *propagate* [0] *force* [0] *merge_blindly* [0] *merge_average* [0] *only_unconnected_single_edges* [0] *quiet* [0]

Merges nodes by tolerance. *name* is the map to merge, *tol* is the tolerance below which to merge. If *pos* and *norm* are non-null triples, they define a plane to limit the node merging to. That is, only nodes on the plane are considered for merging. If *propagate* is selected set, then the merging will continue through the mesh. If *force* is set, then the nodes will be merged regardless of their dimension. If *merge_average* is selected set, then the nodes will be merged by the average. Set *quiet* to 1 for no messages.

ic_uns_merge_node_numbers *nums tol pos* [0 0 0] *norm* [0 0 0] *propagate* [0]

Tries to merge the list of nodes given in a flat TCL list nums, returns the number of nodes merged. For the remaining arguments, see **ic_uns_merge_nodes** above.

ic_uns_min_tri_size *name*

Reports the min triangle size.

ic_uns_move_nodes_exact *name set pos dir csys_name* [""]

Moves nodes.

ic_uns_delete_nodes *name fix_uncovered* [1]

Deletes elements adjacent to nodes in the given map

ic_uns_refine_surface *name dev steps proj tri_only* [0] *edge* [-1] *asp* [-1] *nobound* [0]

Refines triangular/quad/tetra surface mesh.

- **dev**: maximum surface deviation or -1
- **steps**: maximum number of refinement steps
- **proj**: do surface projection if not 0
- **tri_only**: refine triangles only if not 0
- **edge**: maximum edge length or -1
- **asp** : maximum aspect ratio or -1

ic_uns_refine_by_midside_nodes *name all project calc_projection calc_projection_tol*

Refines the mesh using mid-side nodes.

for example, **`ic_uns_refine_by_midside_nodes name all project 1 calc_projection 1 calc_projection_tol 0.01`**

- **name all**. The only option currently available is **name all**, to refine the mesh globally.
- **project** (0 or 1), specifies whether to project the new nodes to the geometry.
- **calc_projection** (0 or 1), when 1, the projection of the node to the marked curves or surfaces of its linear neighbors, and the minimum projection will be calculated.

- **calc_projection_tol** specifies the projection tolerance. The actual node projection will be compared to the minimum projection to see if it is within this specified tolerance. If not, the minimum projection will be used.

ic_uns_insert_tetra *name* [all] *mid* [0]

Inserts tetras.

ic_uns_delete_contents *name*

Deletes the contents of a subset.

ic_uns_set_node_dimension *name how add_bars* [0] *fam_active* [""] *proj_mth* [0] *normal_flag* [0] *dir_vec* [""]

Projects nodes.

ic_uns_get_node_positions *name*

Gets node positions.

ic_uns_set_node_positions *name pos_list*

Sets node positions.

ic_uns_get_element_vertex_positions *name allow_dups*

Gets all the node positions for the elements in this map. This returns a list of {x y z} values, for each node in each element. If *allow_dups* is 1 duplicates are allowed, otherwise the duplicates are filtered out and no guarantee is made about the ordering of the vertex positions.

ic_uns_add_adjacent_tetras *dest src*

Adds adjacent tetras from one map to another.

ic_uns_add_near_node *dest src*

Adds all elements to a map near the nodes in another map

ic_uns_update_family_type *name fams* [__all__] *types* [__all__] *what* [update] *force* [1]

Restricts what is in a subset to a set of families and types.

- **name** is the name of the subset
- **fams** is a list of families to enable, or if preceded by a ! then to disable (if a family is not listed then it is not touched, __all__ means all families, and __same__ means do not touch any)
- **types** is a list of types to enable, or if preceded by a ! then to disable (if a type is not listed then it is not touched, __all__ means all types, and __same__ means do not touch any)
- **what** is update, restrict_to, add, or remove
- **force** is 0 if this command should not touch non-family-type maps.

ic_uns_split_node_non_man *sel_map move_map*

Splits a node non-manifold.

ic_uns_fix_non_man *name*

Fixes some non-manifold vertices.

ic_uns_fix_missing_internal *name fam proj*

Fixes missing internal faces.

ic_uns_fix_uncovered_faces *name fam quiet* [0]

Fixes uncovered faces. Set *quiet* to 1 for no messages.

ic_uns_fix_triangle_boxes *name fam*

Fixes triangle boxes.

ic_uns_fix_volume_orientation *name*

Fixes incorrectly oriented volume elements

ic_uns_fix_subset_by_remeshing *mapname*

Fixes problematic elements (plus two layers) by remeshing the map of given name

ic_uns_scan_plane *mapname what args*

Creates a "scan plane" based on one element. This only works for hexas and prisms ssname set n1 n2 n3 size1 size2 size3 ssname advance n1 n2 n3

ic_uns_duplicate_elements *src dest*

Duplicates elements and their nodes.

ic_uns_move_elements *name args*

Moves an existing mesh subset. The **name** argument gives the subset to operate on.

- **cent** : a list of X Y Z giving the center for rotation, scaling, and mirroring, which could be "centroid"
- **translate** : the X Y Z values for translating the entity
- **rotate** : the number of degrees to rotate the object about the axis given by **rotate_axis**
- **rotate_axis** : the vector giving the axis to rotate about
- **scale** : a scale value, or 3 values giving the factor to scale in X Y and Z
- **mirror** : the axis about which to mirror
- cent, translate, rotate_axis and mirror are defined in the active LCS

ic_uns_subset_create *name* [""] *vis* [0] *copy_from* [""] *pid_colors* [1] *editable* [1]

Creates a subset. The options are:

- **name** : the name to use - default is a uniquely generated one
- **vis** : 1 if the map should be visible
- **copy_from** : if not blank, the name of a map to copy the initial contents of this one from
- **pid_colors** : if 1, display the elements colored by family, otherwise the color parameter of the map is used
- **editable** : 1 if this map should be modifiable

The return value is the name of the map.

ic_uns_subset_create_family_type *fams types empty_ok* [0] *make_visible* [1] *make_explicit* [0]

Creates a temporary subset with a given set of types and families. Either may be __all__. Returns "" if the thing is empty unless empty_ok is 1, in which case it returns the empty map. If make_visible is 1 then it will make this subset visible. If *make_explicit* then it will be promoted to EXPLICIT_LIST

ic_uns_subset_exists *name*

Checks if there is a subset with the given name.

ic_uns_subset_rename *old new*

Renames a subset.

ic_uns_subset_unused_name *pref* [""]

Returns an unused subset name with the given prefix. Note that this gives names unique for both geometry and mesh.

ic_uns_subset_color *name color*

Changes the color on a subset. Disable pid colors.

ic_uns_subset_copy *dest src etype* [""]

Copies from one subset to another. If etype is 0, 1, 2, 3, or a list of those, then copy only elements of that dimension. Also etype can be node or edge which will pick nodes or bars.

ic_uns_subset_copy_verts *dest src*

Copies vertices from one subset to another.

ic_uns_subset_copy_if_not_explicit *name*

If this map is not explicit, make a copy of it and make sure that it is.

ic_uns_invert_subset *name refmap* [all]

Inverts the contents of a subset.

ic_uns_subset_into_connected_regions *which*

Decomposes the given subset *which* into a set of subsets, each being a region of connected elements. The given subset is unmodified. The return value is a list of names of the new subsets.

ic_uns_subset_surface_boundary *sname*

Returns the name of a subset containing the surface boundary elements of the given surface name *sname*. Returns a nullstring in case of an error.

ic_uns_subset_get_vertices_on_curves *curve_names*

Returns the name of a subset containing the ordered vertices of the bars along the given set of curves *curve_names*. Returns a null-string in case of an error.

ic_uns_subset_get_current

Returns the name of the current visible map - maybe a subset and maybe selected.

ic_uns_subset_set_current *name*

Sets the current visible map.

ic_uns_subset_delete *name*

Deletes a subset.

ic_uns_subset_configure name args

Configures a subset

ic_uns_subset_configure_toggle *name args*

Configures a subset by toggling the given arguments.

ic_uns_subset_configure_get *name param*

Gets the value for a subset's given configuration parameters.

ic_uns_subset_visible *name vis*

Makes a subset visible (or not).

ic_uns_subset_list_visible *except* [""]

Lists the names of the visible subsets.

ic_uns_subset_is_cut *name*

Checks to see if a subset has the cut flag enabled.

ic_uns_subset_count_elements *name type* [MIXED]

Counts the number of elements in a subset.

ic_uns_subset_count_nodes *name*

Counts the number of nodes in a subset.

ic_uns_subset_average_edge_length *name*

Gets the average edge length in a subset.

ic_uns_subset_average_surface_deviation *name*

Gets the average surface deviation in a subset.

ic_uns_subset_list_families *name*

Lists the families that appear in a subset. Each family appears only once.

ic_uns_subset_list_volume_families *name*

Lists the volume families that appear in a subset. Each family appears only once.

ic_uns_subset_families_used *name parts*

Returns the families or parts that are used in this subset. If there are nodes in the subset then the elements attached to those nodes will also be included.

ic_uns_subset_copy_only_families *from fam ispart*

Copies only the elements in a specified family or part into a new subset.

ic_uns_subset_list_elements *name*

Lists all the element data for a subset in the following format: {{eltype elnum ext_elnum {elnode1 elnode2 ...} fam_name} {eltype elnum ext_elnum {elnode1 elnode2 ...} fam_name} ... {VERTEX vnum {x y z} dimension extvnum} {VERTEX vnum {x y z} dimension extvnum} ...} where element vertices specified explicitly in the subset (not elements of type NODE) are at the end of the list.

ic_uns_subset_nodes_are_periodic *name*

Checks if any nodes in this subset are periodic.

ic_uns_subset_list_types *name*

Lists the element types that appear in a subset. Each type appears only once.

ic_uns_subset_list *only_user* [0]

Lists all the subsets that are editable. If *only_user* is 1 then skip the ones that begin with uns_sel_ or CONTACT.

ic_uns_map_list *only_vis pat* [*]

Lists all the maps (optionally just the visible ones). If *pat* is given then return those whose names match.

ic_uns_subset_clear *name*

Removes all entities from a subset.

ic_uns_subset_add_from *dest src*

Adds entities from one subset to another.

ic_uns_subset_subtract_from *dest src*

Removes entities in one subset from another.

ic_uns_subset_add_number *name nums rel type*

Adds a list of elements by numbers. *rel* = 0 : internal within the given type, *rel* = 1 : internal within all types, rel = 2 : external. Note that the *nums* can also be ranges of the form N1-N2.

ic_uns_subset_add_node_numbers *name nums extern* [0]

Adds a list of nodes by numbers. If extern is 1 use external numbers otherwise internal. Note that the nums can also be ranges of the form N1-N2.

ic_uns_subset_list_node_numbers *name extern* [0]

Lists the unique internal (external) node numbers of the subset. If extern is 1 use external numbers otherwise internal.

ic_uns_subset_add_region *name p1 p2 part dims* [0 1 2 3]

Adds elements in a region.

ic_uns_subset_add_near_pos *name pos dims* [0 1 2 3] *max_dim* [""]

Adds elements near a position.

ic_uns_subset_add_near_nodes *name nodes ext* [0]

Adds elements near a set of nodes.

ic_uns_subset_add_layer *name nlayers fams types*

Adds a layer. If *fams* = active then it means active families, all means all families. types = active means active types, all means all types, surf means all surface types.

ic_uns_subset_remove_layer *name nlayers types*

Removes a layer. If *fams* = active then it means active families, all means all families. types = active means active types, all means all types, surf means all surface types.

ic_uns_subset_add_attached *name src how angle* [0]

Adds attached elements.

ic_uns_subset_get_elements_attached_to_subset *src_map des_families* [""] *des_types* [""]

Gets a map of those elements of type *des_types* in *des_families* that are attached to the subset with name *src_map*.

ic_uns_subset_remove_normal *name norm*

Removes elements with a specific normal.

ic_uns_subset_remove_manifold_elements *name*

Removes all surface elements that are connected to one other surface on all sides.

ic_uns_subset_get_cutspeed *name*

Restricts a subset to its cut. Note that the point and normal given here are in the **global** coordinate system. This returns the number of elements displayed in the cut.

ic_uns_subset_bbox *name no_cut* [0]

Returns the bounding box of a subset. If *no_cut* is 1 then it will be the bounding box with no plane cut taken into account.

ic_uns_subset_normal *name*

Returns the average normal of a subset. This skips any non-2D elements.

ic_uns_subset_is_editable *name*

Checks if this subset is editable (i.e. not selected or all).

ic_uns_min_metric *crit* [Quality] *parts* [""] *types* [""]

Returns the min value (max value in case of Volume change) of a quality metric (and "" in case of an error).

For example, set worst [**ic_uns_min_metric** "SPECIFIED_METRIC" "SPECIFIED_PARTS" "SPECIFIED_TYPES"]

Default for SPECIFIED_METRIC is "Quality". Default for SPECIFIED_PARTS is all existing parts. Default for SPECIFIED_TYPES is all existing types (but skips NODE, LINE and BAR elements). Each of them can be empty.

ic_uns_create_selection_subset *color_first_different color* [""]

Creates a temporary subset for selection. This one will not show up in the subset list. If *color_first_different* is 1 then the first node will get a special color. All the other nodes and elements are drawn in the specified **color**.

ic_uns_create_selection_edgelist *on*

Creates or deletes a temporary edge list for selection (depending on whether *on* is 1 or 0).

ic_uns_subset_add_pick_polygon *what name poly vect from_names partial convex etype* [""]

Adds elements in a screen polygon area to the subset, *what* can be one of edge, node, element, or a list of one or more of 0d, 1d, 2d, 3d.

ic_uns_subset_add_pick_circle *what name vect cent radp from_names partial etype* [""]

Adds elements in a screen circular area to the subset, *what* can be one of edge, node, element, or a list of one or more of 0d, 1d, 2d, 3d.

ic_uns_subset_add_pick_single *what name pt vec from_names*

Adds a picked element or whatever to the subset, *what* can be one of edge, node, element, or a list of one or more of 0d, 1d, 2d, 3d. Returns the element number that was picked, or "" if none was picked.

ic_uns_subset_add_flood *what name from*

Adds connected boundary edges to the subset.

ic_uns_subset_add_remove_last *name*

Removes the last bunch of entities added.

ic_uns_create_vertex *name pt dim*

Creates a new vertex.

ic_uns_subset_make_all *name*

Makes the current subset an all-subset.

ic_uns_subset_make_families *name fams*

Makes the current subset a family subset.

ic_uns_subset_add_families_and_types *name fams types*

Makes the current subset have a certain set of families and types.

ic_unstruct_make_families_exist *fams*

Makes new unstruct families, returning the number of those which are new.

ic_uns_uniqify *name*

Makes sure that the entities in this subset are unique.

ic_uns_node_position_stack *name what*

Pushes, pops, or clears node positions on the stack for undo in interactive move.

ic_uns_node_position_drag *name startpt pt vec spl curvis allow_invert fam_active*

Drags nodes interactively. This projects to the surfaces, curves, etc. as appropriate for the nodes, if *fam_active* is a list of families. The strings all or none can also be given for *fam_active*.

ic_uns_bandwidth *name iters type*

Bandwidth and profile reduction by the reverse Cuthill-McKee (RCM) algorithm.

- **iters**: maximum number of renumbering steps
- **type**: reduce profile if 0, reduce bandwidth if 1

ic_uns_renumber_nodes *which start skip_0* [0] *dir* [0 0 0]

Resets external node numbers, starting with *start*

ic_uns_renumber_elements *which start skip_0* [0] *dir* [0 0 0]

Resets external element numbers, starting with *start*

ic_uns_renumber_all_nodes *start skip_0* [0]

Resets external node numbers for all elements, starting with *start*, less memory usage compared to **ic_uns_renumber_nodes all**.

ic_uns_renumber_all_elements *start skip_0* [0]

Resets external element numbers for all elements, starting with *start*, less memory usage compared to **ic_uns_renumber_elements all**.

ic_uns_subset_elements_range *name*

Lists external start and end node/element numbers

ic_uns_subset_move_part_elements *name oldpart newpart*

Moves elements in a subset from one part (oldpart) to another (newpart) Note that old part must exist, new part will be created if not existing

ic_uns_create_diagnostic_subset *vis* [1]

Creates a temporary subset for diagnostics.

ic_uns_create_diagnostic_edgelist *on*

Creates a temporary edge list for diagnostics.

ic_uns_convert_edgelist_to_bars *name family*

Converts the current diagnostic edgelist to bar elements and put them in the named map. They are given the specified family.

ic_uns_diagnostic *args*

Runs diagnostics on the loaded unstructured mesh. The arguments come in keyword value pairs. The possible options are:

- **diag_type***what* : the type of diagnostic. This can be one or more of the following:
  - **duplicate** : check for duplicate elements
  - **single** : check for single-element edges
  - **single_2** : check for 2-single-edged elements
  - **multiple** : check for multiple-element edges
  - **single_multiple** : check for single/multiple edged elements

  - **uncovered** : check for uncovered volume faces
  - **missing_internal** : check for missing internal faces
  - **triangle_box** : check for triangle boxes
  - **nmanvert** : check for nonmanifold vertices
  - **vol_orient** : check for mis-oriented volume elements
  - **surf_orient** : check for mis-oriented surface elements
  - **disconnected_vert** : check for disconnected vertices
  - **3_surface_node_internal_faces** : check for spanning edges
  - **periodic_problem** : check for problems with periodicity
  - **overlap** : check for overlapping elements in the mesh
  - **delaunay** : check for Delaunay violations in the mesh
  - **standalone** : check for shells not connected to any volume
  - **metric** : apply one of the quality metrics
  - **all** : do all of the above

- **type***typename* : look at elements of this type, which can be TRI_3 or QUAD_4. If no type name is given then all types are selected.
- **family***famname* : check elements of this family. This argument can be given more than once for multiple families. If no family is given then all families are selected.
- **fams***famnames* : check elements of the named families. This is just like the family argument except you can give multiple families. If no family is given then all families are selected.
- **fix***on* : fix the elements that fail the check, if possible (default is 0)
- **fix_fam***family* : if fix is selected and new elements have to be created, put them in this family
- **fix_project***on* : if fix is selected and new nodes are created, project them to the geometry. This is applicable only for for the **missing_internal** diagnostic type.
- **minval***val* : the minimum acceptable value for a quality check (default is 0)
- **maxval***val* : the maximum acceptable value for a quality check (default is to not check this)
- **metric***what* : which quality metric to use (default is Quality)
- **disp_elems***on* : if **on** is 1 then display the problem elements and wait for user confirmation before proceeding. If it is 2 then the elements will remain displayed until cleared using **ic_clear_diag_display**. If **fix** was given then only those elements that could not be fixed will be displayed. The default is 0.

ic_uns_get_diagnotic_info *diagnostic param*

Gets the value for a particular *param* for a particular *diagnostic*

ic_uns_metric *name* [all] *type* [Quality] *args* [""]

Runs diagnostics on the loaded unstructured mesh. There may be additional optional arguments:

- **prism_warp_weight***value* : a value between 0 and 1 that controls the relative weighting of warp and aspect ratio for prism quality

Returns a list in the following order: number of elements with this diagnostic, number of elements for which this diagnostic is undefined, not used, worst element quality with this diagnostic, best element quality with this diagnostic, used to calculate the mean quality

ic_uns_n_els_in_metric_range *fams types diag from to*

Gets the number of elements in the given *fams* of the given *types* which fall between *to* and *from* when evaluated by the given metric *diag*

ic_uns_subset_n_els_in_metric_range *map diag from to*

Gets the number of elements in the given *map* which fall between *to* and *from* when evaluated by the given metric *diag*.

ic_uns_set_metric_weight *metric eltype args*

Sets weights (min, ideal, max) for a given metric for *eltype*

ic_uns_set_metric_weights *list reset* [1]

Sets weights for a list of metrics such that each element of the list matches the arguments to **ic_uns_set_metric_weight**, i.e. {{metric eltype weights} ...}

ic_uns_get_metric_names

Gets a list of the metric names that are understood

ic_uns_get_metric_weights *metric*

Returns a list of the current custom metric defs, e.g: { {{Skew} TRI_3 acc_min 0 des_min 0 ideal 0 des_max 45 acc_max 60} \ {{Skew} QUAD_4 acc_min 0 des_min 0 ideal 0 des_max 45 acc_max 60} \ ... \ }

ic_uns_print_diagnostic_quality_key *metric_name*

Prints the key used for the coloring of elements in the display.

ic_uns_histogram *name min max nbars*

Uses the diagnostic data for a histogram

ic_uns_set_with_diagnostic *name from intervals*

Sets the elements in a subset via histogram intervals.

ic_uns_diagnostics_update

Updates the diagnostic values on the mesh, and return the number of elements on which the diagnostic was updated. If no diagnostic has been run, this procedure will return 0.

ic_clear_diag_display

If any diagnostic graphics are left on the screen after an **ic_uns_diagnostic** call with **disp_elems** set to 2, erase them.

ic_uns_mesh_distribution *p1 p2 what* [0]

Gets the mesh distribution along a line in the mesh

ic_uns_show_selected *type names on color* [""] *force_all_types* [0]

Displays some unstructured elements selected or not

ic_uns_reset_selected

Resets all selection display.

ic_uns_make_periodic *name*

Makes the nodes in a subset periodic.

ic_uns_make_periodic_from_element_pair *e1 e2*

Makes the nodes of two faces periodic from a given element pair

ic_uns_make_auto_periodic *src dest*

Makes the elements in a subset periodic

ic_uns_remove_periodic *name*

Makes the nodes in a subset non periodic.

ic_flood_fill *args* [""] *keep_orfn* [0]

Performs the flood-fill operation of cutter

med crashes if no proj model - Bruce

This clashes with geom undo - Ganesan

If periodic with pentas or pyras, the surface mesh may be lost - Bruce

ic_flood_fill_mesh *keep_orfn* [0] *only_mat* [0]

Flood fills a mesh. Volumes need to be enclosed by surface elements.

ic_auto_orphan

Automatically creates orphan regions.

ic_uns_quad_remesh *map args*

Runs the quad mesher to fix holes, etc.

ic_uns_force_node_loc *ubset_name location method* [move]

Makes a node exist at location in the subset *subset_name* by one of these methods: move.

ic_get_disconnected_vertices

Returns a list such that the first element is the number of disconnected nodes found in the current unstructured mesh, and the second element is the subset containing those nodes.

ic_delete_disconnected_vertices

Deletes the disconnected vertices in the given subset, or if the map is not given, in the current mesh, and returns the number of disconnected vertices deleted.

ic_uns_build_mesh_topo *which args*

Builds the topology information from the mesh subset given *which*. It is fairly analogous to **ic_geo_build_topo**. It will also do call: **ic_uns_set_node_dimension** all reset after running. Arguments *args* are:

- **-angle***angle* : the angle for extraction (degrees)
- **-smart** : according to angle, choose options which seem best
- **-new_element_family***family_name* : family name for new elements
- **-uncovered_element_family***family_name* : family name for any uncovered faces
- **-save_old** : put removed elements in ORFN (else delete)
- options:
- **-remove_points (or -filter_points)** : try removing points at curve intersections, the difference between tangents of which is below specified angle
- **-remove_bars (or -filter_bars)** : try removing bars
- **-create_points** : try add points at bar intersections, the difference between tangents of which is below specified angle
- **-create_bars_single_edges** : create bars on single edges in mesh
- **-create_bars_multiple_edges** : create bars on multiple edges in mesh
- **-create_bars_double_edges** : create bars on double edges in mesh (using -angle)

ic_uns_offset_mesh *args*

Offsets the mesh by distance. Right now it only works on the whole mesh.

- **dir***direction* : elements should be offset in this direction. It can take any one of the following strings: normal - Normal to the current element normals "$x $y $z" - Arbitrary direction [+|-][x|y|z] - e.g. +x: offset with respect to the current element normal in the given +x direction.
- **dist***distance* : elements should grow in this direction. Example: ic_uns_offset_mesh map All dir +x dist 2

ic_uns_internal_wall_elements_get

Returns a subset with internal wall elements.

ic_uns_internal_wall_families_get

Returns a list of families with internal walls.

ic_uns_internal_wall_families_reorient *fams*

Reorients the internal walls of given families to have consistent normals. Returns the number of element orientations changed.

ic_uns_spectral_elements *name file order law*

Calculates spectral edges.

ic_validate_families

Validates the match between pids in proj and uns. For debugging.

ic_uns_load_temporary *file params* [-color white]

Creates a temporary mesh.

ic_uns_unload_temporary *name*

Deletes a temporary mesh.

ic_uns_get_thickness *name*

Gets the thickness associated with an element. This returns a list of lists, for each element: num thick1 thick2 thick3 [thick4]

ic_uns_contact_sets *dist contactpartlist* [""] *debug* [0]

Calculates target families based on proximity factor Returns a list of maps for the contact locations.

ic_uns_immutable *name what*

Sets an element(s) as immutable

ic_uns_isimmutable *name*

Checks if element(s) is/are immutable

ic_unstruct_grad_smooth *iter_sf iter_vl alpha_sf alpha_vl crit_avr_cha_sf* [0.0001] *crit_avr_cha_vl* [0.0001] *n_fix_layer* [0] *sfit* [0] *fams_attr* [""] *rebunch_edges* [0] *family_edges* [""] *edges* [""] *families* [all] *update_mesh_step* [0]

Gradient smoothing

ic_unstruct_elliptical_smooth *iter_sf iter_vl gexpr_sf gexpr_vl stabilize_sf stabilize_vl ortho_distance_sf ortho_distance_vl stype fix n_fix_layer fams_attr rebunch_edges sfit family_edges edges* [""] *families* [all] *treat_unstruct* [0] *kpg* [0] *niter_post* [0] *limit_post* [0.2]

Elliptical smoothing.

ic_uns_thickness_exists_mesh

Checks if thickness is stored in the mesh.

ic_uns_get_thickness_node *name*

Gets the thickness associated with an element-node. This returns a list of lists, for each element-node: num thick

ic_uns_surface_volume *name*

Volume of surface elements for a part, sum of surface area of element * averaged thickness

ic_uns_bcfield_define *name def*

Defines a new bcfield, or change the default value for the field.

ic_uns_bcfield_exists *name*

Checks if a bcfield exists.

ic_uns_bcfield_list

Lists the available bcfields.

ic_uns_bcfield_get *name mapn def_ok*

Gets bcfield values.

ic_uns_bcfield_default *name val* [""]

Gets or sets bcfield default value.

ic_uns_bcfield_set *name mapn val*

Sets bcfield values.

ic_uns_bcfield_rename *name newname*

Renames a bcfield.

ic_uns_subset_update_bcfield_color_map *bcfield subset*

Sets up a color map for this subset that is appropriate to the given bcfield. There has to be just one of these visible at a time.

ic_uns_set_part_bar_orientations *parts args*

Sets bar orientations on a given set of parts.

ic_uns_enforce_bar_direction *name*

Makes sure the bars in a specific map are all oriented such that no node has more than one bar pointing into it, if possible. Return value is the number of nodes that are still violating the constraint.

ic_uns_mesher_license *get*

Checks out or gives back a mesher license appropriate to the current mesh. Returns an error string if there was a problem.

ic_uns_associate_geometry

Para mesh.

ic_uns_split_edges_propagate *which par* [0.5]

Splits edges in subset and propagates.

ic_uns_split *p0 p1 args* [.5]

Splits an edge in the mesh

ic_uns_convert_tet2_hexcore *fams* [""] *fill_holes* [0] *refinement* [1] *bbox* [""] *csys* [global] *conformal* [0] *use_tg_tri2tet* [0]

Converts tetra volume mesh to Hexa-core mesh.

ic_uns_write_node_list *map file scale* [1]

Writes out the nodes of a subset in a simple format.

ic_uns_write_elem_list *map file*

Writes out the elements of a subset in a simple format.

ic_uns_import_bcfield *name file on_nodes colnum defval*

Reads in per-element or per-node result domain data. The file contains 2 or more columns. The first is the element or node number, and *colnum* selects which of the following ones is loaded, starting with 1. The name is something like Nastran:OR:X1 and will be available as a distributed attribute in MED. If the attribute did not already exist it will be created and given the specified default value. If *on_nodes* is 1 then the numbers in the file are external vertex numbers and the bc values will be attached to the NODE elements on those vertices. If a vertex is referenced that does not have a NODE element it will be skipped (for now).

ic_uns_export_bcfield *name file on_nodes*

Exports BC values in the same format. If *on_nodes* is given then the nodes of the given elements are written out with the values for those elements, and duplicated nodes are written more than once (possibly with different values).

ic_list_volume_families

Returns the volume parts.

ic_uns_count_nodes_in_families *fams*

Counts the number of nodes in each family.

ic_uns_mark_enclosed_elements *name surfs matlpnts only_vol* [0]

Marks elements enclosed by surfaces with the material points.

ic_uns_bounding_box

Returns the bounding box of the mesh.

ic_uns_count_elements_in_family *fams*

Counts elements in families.

ic_uns_subset_from_part *parts verts* [0]

Creates a subset from part names. If *verts* is given, vertices are added to the subset if true.

ic_uns_get_periodic_parts

Returns the periodic parts, or nothing if no periodic parts exist.

ic_uns_count_periodic_nodes *name*

Counts the periodic nodes.

Cartesian Mesh Editing and Modification Functions

ic_cart_is_loaded

These functions are used to manipulate the **cart_file** data which is used by the **Global** mesher. Note that the meshes output by **Global** global can be cylindrical in addition to purely Cartesian. The positions in X, Y, and Z appear in the cart file, together with information on the coordinate system.

In addition to the explicit stations, one or more *ranges* can be stored in the cart file. These ranges specify a distribution of stations. When the list of stations is requested for a direction, or the file saved or displayed, these ranges will be evaluated to give the result. Each range contains the following parameters:

- *start*: the starting position for the range
- *end*: the ending position for the range
- *type*: one of the following values:

  - *uniform*: the stations are uniformly spaced in the region
  - *geo_start*: the spacing is geometric, from the start point
  - *geo_end*: the spacing is geometric, from the ending point back towards the start
  - *geo_both*: the spacing is geometric in both directions. **Currently unimplemented**
  - *explicit*: the specific list of stations is given. **This option is currently unavailable from the ic_ functions**.

- *count*: the number of stations for uniform spacing or an explicit list
- *start_h*: the starting spacing if the type is *geo_start* or *geo_both*
- *start_r*: the starting ratio between adjacent stations if the type is *geo_start* or *geo_both*.
- *end_r*: the ending spacing if the type is *geo_end* or *geo_both*.
- *end_r*: the ending ratio between adjacent stations if the type is *geo_end* or *geo_both*.
- *comb*: how to combine this range with other ranges. If *comb* is 0 then the new stuff will wipe out all the old stuff in the regions where it covers it, if 1 then the old and new stuff gets merged, and if 2 then the new stuff will be used only in regions that are NOT inside a previous range where we also had a value of 2, but the old stuff will be always pulled in. However the new stuff will always contribute its starting and ending points no matter what.
- *max_h*: the maximum spacing for the entire range
- *active*: 1 if this range is to be used, 0 if it's inactive

Checks if cart file information has been loaded into the program.

ic_cart_is_modified *set* [""]

Checks if the cart file info has been modified.

ic_cart_get_bbox

Returns the bounding box of the cart file stations. The return value is a list of 2 triples: e.g. {0 0 0} {1 2 3}

ic_cart_unload

Unloads the current cart file data.

ic_cart_load *file*

Loads cart data from the given file name. It is not initially visible.

ic_cart_save *file*

Writes out cart data to the given file.

ic_cart_initialize *from_proj bbox* [""]

Creates an empty cart file with the boundary of the current viewable stuff as the stations. There will be 2 stations in each of the 3 directions which are at the min and max of the visible area. If the *from_proj* argument is 1, and a tetin file is loaded, then instead the geometry data will be used and a station will be created at each "key point" of the geometry.

ic_cart_stats

Return stats for the cart file. 4 integers are returned: the number of stations in each direction, followed by the number of nodes (i.e. the product of these three values).

ic_cart_visible *on*

Enables/disables the cart file visibility.

ic_cart_grid_type

Returns the type of the loaded cart file: either *cartesian* or *cylindrical*.

ic_cart_set_grid_type *type*

Set the type of the loaded cart file. *type* must be either *cartesian* or *cylindrical*.

ic_cart_add_screen *pt vec*

Add one station to the cart file, based on the vector defined by *pt* and *vec*. These are the values determined by screen selection.

ic_cart_select_screen *pt vec*

Select stations in the cart file via the screen, based on the vector defined by *pt* and *vec*.

ic_cart_delete_selected

Delete the selected stations in the cart file, which have been previously defined using **ic_cart_select_screen**.

ic_cart_list_stations *dir*

List the stations in a direction. *dir* should be either 0, 1, or 2.

ic_cart_set_stations *dir stats*

Set the stations in a direction. *dir* should be either 0, 1, or 2. *stats* is a list of floating point numbers which are the stations in that direction.

ic_cart_list_ranges *dir*

List the ranges in a direction. *dir* should be either 0, 1, or 2. The return value is a list of ranges, where each range has the format:

start end type numpts start_h start_r end_h end_r comb max_h active

The meaning of these parameters is described at the start of this section.

ic_cart_set_ranges *dir ranges*

Set the ranges in a direction. The *ranges* argument is a list of ranges as described above.

ic_cart_get_axes

Get the axes for the cart file. This returns a list of 4 triples: *origin axis0 axis1 axis2*

ic_cart_set_axes *origin axis0 axis1 axis2*

Set the axes for the cart file. The arguments are the origin and the 3 coordinate axes, each of which is a list of 3 numbers.

ic_cart_is_from_ddn

Checks if the cart file came from DDN. This matters because the cart files from DDN are written out together with the geometry pre-transformed into the given coordinate system, but other cart files are not.

Miscellaneous Mesh Editor Functions

ic_rename_family *old new*

Renames the specified family in the tetin database and the unstructured mesh (domain file).

ic_print *args*

Write hardcopy or image file output. The arguments are:

- **format***form*: one of:
  - **ps**
  - **ppm**
  - **x**
  - **tiff**
  - **gif**
  - **jpeg**
  - **rgb**
  - **win**
- **generate_new** generate a new filename based on the **outfile** argument
- **invert***on* : swap black and white
- **jpeg_quality***qual* : the quality factor for JPEG, between 0 and 100
- **landscape***on*: if 1 then swap the X and Y axes
- **outfile***filename*: the file to create (without the suffix which is automatically added)

- **ps_color**_opt_: either color, gray, or mono
- **ps_direct**_on_: if 1 then write a vector-based ps output, otherwise do raster
- **ps_frame**_on_: if 1 draw a frame around the ps output
- **ps_label**_text_: the label to put at the bottom right of the postscript output
- **ps_title**_text_: the title to put at the top of the postscript output
- **scale**_val_: scale factor
- **send_to_printer**_on_: if 1 then the postscript file will be printed using lp

ic_view_zvector

Returns the current vector into the screen. Returns 0 0 0 in batch mode.

ic_comment _text_

Introduce a comment line into script file, for example, `ic_comment "Why was this command added to the API?"` The command has no effect on program execution.

ic_get_n_processors

Returns the number of processors.

Output Functions

ic_create_output *solver domain args*

Write an input for a solver. *solver* is the name of the solver. Currently supported solvers are:

- **"Ansys Fluent"**
- **STAR-CD**
- **CFX-4**

domain is the full pathname to the domain to be used. In the case of a structured mesh this must be a list of the individual structured domain files.

The rest of the arguments are a series of name value pairs. The names depend on the solver being used.

- **"Ansys Fluent"**
  - **outfile***filename* (required) : the .msh file to write
  - **bocofile***filename* (required) : the family_boco or boco file to use
  - **dim2d***on* (default 0) : if 1 write a 2-D mesh
  - **set_tol***on* (default 0) : if 1 specify a tolerance
  - **tol***val* (default 0.0001) : if **set_tol** was given use this tolerance for the periodic node section
  - **result***on* (default 0) : if 1 create a .dat file also
- **STAR-CD**
  - **outputfile***filename* (required) : the prefix of the files to write
  - **bocofile***filename* (required) : the family_boco or boco file to use
  - **nstart***val* (default 5) : starting ICTID number
  - **unformat_flag***val* (default 0) : if 1 write an unformatted output, otherwise formatted
  - **no_node***val* (default 0) : do not write a node file
  - **no_elem***val* (default 0) : do not write an elem file
  - **no_bread***val* (default 0) : do not write a bread file
  - **no_shell***val* (default 1) : do not write a shell file

- **version***val* (default 3100) : what version of starcd (possible values are 3100 and 3050)
- **laminar***val* (default 0) : 1 if laminar flow, 0 if turbulent
- **all_bnd***val* (default 0) : if 1 write all boundaries, if 0 only those with a bc assigned
- **inlet_model***val* (default keps) : either keps or mixl
- **scale***val* (default 0) : if 1 then do scaling
- **x***val* (default 1.0) : the x scaling value
- **y***val* (default 1.0) : the y scaling value
- **z***val* (default 1.0) : the z scaling value

• **CFX-4**

- **outputfile***filename* (required) : the geometry file to write
- **bocofile***filename* (required) : the family_boco or boco file to use
- **topofile***filename* (required) : the topology file to use
- **family_topo***filename* (required) : the family_topo to use
- **topo***on* (default 0) : create a CFX command file
- **scale***on* (default 0) : scale geometry by the given factors
- **x***val* (default 1) : X factor for scaling
- **y***val* (default 1) : Y factor for scaling
- **z***val* (default 1) : Z factor for scaling
- **thickness***val* (default 1) : thickness value for 2-D planar grids

ic_export_mesh *args*

Export mesh.

ic_import_nastran *prog outfile rbe3 bars shell projname*

Runs import mesh Nastran command.

ic_export_nastran *nas_file domain family_boco solver_params large_format vol_v shell_v bar_v*

For internal use only.

ic_run_stars_result2df *srf_file domain_file mesh_info type*

Runs Star's result to domain file translator.

Boundary Condition Editing and Modification Functions

ic_boco_load *file*

Loads in the boundary condition data from the *file*.

ic_boco_load_atr *file*

Loads in the attribute data from the *file*.

ic_boco_save *file*

Saves the current boundary condition data to the given file name.

ic_boco_save_atr *file ss* [""]

Saves a new format bc (.atr) file.

ic_boco_is_loaded

Checks if a family boco file has been loaded.

ic_boco_is_modified

Checks if the current boco data has been modified since the last save.

ic_boco_set_modified

Forces the current boco data to be dirty.

ic_boco_unload

Unloads the current family boco data.

ic_empty_boco

Creates an empty boco database.

ic_boco_solver *solver* [""]

Sets the default solver for an existing boco database. If no solver is given, then it returns the current solver setting.

ic_boco_get *fam bctypes* [""] *not_bctypes* [""]

Gets the boundary conditions associated with the given family. They are returned as a list, each element is one boco. Each boco element is a list where the first element is 0 or 1 whether the data is active or not, the second is the type, such as WALL or INT, and the rest are the parameters. For information about what the available types and parameters are, see the **Solver Parameters and Boundary Condition Data** section for the solver you are interested in. If the *bctypes* argument is given then only the matching boundary conditions are returned.

ic_get_boco *fam*

This is the same as **ic_boco_get** (for backward compatibility only).

ic_boco_set *fam bocos*

Sets the boundary conditions associated with the given family. They must be given as a list, each element is one boco. Each boco element is a list where the first element is 0 or 1 whether the data is active or not, the second is the type, such as WALL or INT, and the rest are the parameters. For information about what the available types and parameters are, see the **Solver Parameters and Boundary Condition Data** section for the solver you are interested in.

ic_boco_append *fam nbcs*

Appends some boundary conditions.

ic_boco_modify *fam obc nbc*

Changes one boundary condition.

ic_boco_delete_bctypes *fam bctypes*

Deletes boundary conditions with a given set of types from a family.

ic_set_boco *fam bocos*

This is the same as **ic_boco_get** (for backward compatibility only).

ic_boco_delete_all_bctype *bctype*

Deletes all boundary conditions on all families of a given type.

ic_boco_bctypes_used_by_families *which* [""]

Returns a list of *bctypefamlistbctypefamlist* ... pairs If *which* is "" then all BCs are returned, otherwise only those that show up in the list. A family could possibly appear twice in the *famlist* if it has the BC twice.

ic_boco_list_with_bctype *bcargs exclude_fam_pattern* [""]

Lists all families and BCs that have a BC of a given type. If *bctype* is a list then it will match the second, etc argument. Glob characters are allowed here. If *exclude_fam_pattern* is non-"" skip families that match that pattern. Generally this will be *:* .

ic_boco_classify_by_bctype *bctype exclude_fam_pattern* [""]

Lists the families by *bctype*.

unused__ic_boco_groups_with_bctype *bcargs exclude* [*:*]

A convenience function that just returns the group names with a specific set of bc args For example
`ic_boco_groups_with_bctype [list FORCE_NAME ff1]`

ic_boco_list_bcargs_for_bctypes *types pos*

Gets all the possible values for an argument at a specific position for a list of *bctypes*. This is used to get a list of load sets.

unused__ic_boco_get_with_matching_name *t1s t2 v2*

Returns the BCs of the types t1s, which are in families whose BC of type t2 has value v2. (e.g.,. *t1s* might be {BR BT}, *t2* might be CONSTRAINT_NAME, and *v2* might be "foo 123". Used for ai*env. Note that this automatically *excludes* families with : in them.

unused__ic_boco_get_unused_bcarg *bctype prefix*

Gets an unused name for a given BC type.

ic_boco_get_unused_group *prefix avoid* [""]

Gets an unused name with a given prefix. Avoid the ones given in the second argument

unused__ic_boco_replace_with_matching_name *newbcs t2 v2 replace_these* [""]

Finds all families that have a *bctype* named *t2* with a value *v2* and replaces the existing BCs that have types in *newbcs* with the new ones. (Ie.g.,. *t1s* might be {{1 BR 1 2 3} ...}, *t2* might be CONSTRAINT_NAME, and *v2* might be "foo 123". Used for ai*env. Note that this automatically *includes* families with : in them. If a value is given for *replace_these* it should be a list of *bctypes* to get rid of, else the types are taken from *newbcs*. Returns the families that were so modified.

ic_boco_replace_in_part *newbcs part replace_these* [""]

Same as above but does it for all families in a part.

ic_boco_replace_in_group *newbcs group replace_these* [""]

Same as above but does it for all families in a group.

ic_boco_add *fam bocos*

Adds some BCs to a family.

ic_boco_replace *fam bcarg newbcs*

Replaces some BCs for a family.

ic_boco_get_for_tetin

Returns the current default *family_boco* file which was loaded (if any).

ic_boco_reload

Reloads the family boco file which has already been loaded (if any).

ic_boco_list_families *pat* [""]

Lists the current families which have boundary conditions associated with them. A pattern may be given that limits the families to those matching the pattern, as part of a :-separated list. *pat* may also be !: which means no : characters.

ic_boco_list_families_only

Changes the *family_boco* file that is associated with the tetin file.

ic_boco_fam_dim *fam all_dim* [0]

Returns the dimension of the family.

Note:

See the next function. If there is more than one type of element in the family then the return value will be *mixed* unless the *all_dim* parameter is set to 1

ic_boco_parts_dim *parts*

Returns the dimensions of the part.

ic_boco_reset_fam_dim

Warning: hack ...

If you call **ic_boco_fam_dim**, then you change something, and then you call it again, you will get the old data. This is because the **list_families_and_dimensions_and_sides** function is too slow so the results have to be cached. As a temporary workaround, call **ic_boco_reset_fam_dim** to clear out the old data.

ic_boco_clear_icons

Deletes all existing BC icons.

ic_boco_rename_family *old new*

Renames a family - this has to move all the icons

ic_boco_add_icon *type fam icontype locations scale maxnum color label params vis only_dims*

Creates a new BC icon for a given family. The arguments are:

- *type* : the kind of BC, such as "temperature", "pressure", etc. If another icon of the same type and family is already there it will be deleted. In general this could be the *bctype* of the BC that this icon represents.
- *fam*: the family name.
- *icontype*: the shape of the icon to draw. These are listed below.
- *locations*: the XYZ locations of where the icons should be drawn. If this list is empty then they will be automatically computed. This list can also be a geometry type, "geometry" or "unsmap". That restricts the automatic computation to that object.
- *scale*: a size scale for the icons. Default is 1.0
- *maxnum*: the maximum number of icons to draw. If there are more nodes or elements than this value then just one icon will be drawn at the centroid of the family. 0 means no limits.
- *color*: the color to use for the icon.
- *params*: depends on the *icontype* value as described below.

The different icon shape types are as follows:

- *arrow*: draw an arrow. The params are:
  - *direction*: an XYZ triple that gives the arrow's direction
  - *nheads_end*: how many heads to draw at the end. If this value is 0 then no head will be drawn. If it is -1 then an X will be drawn.
  - *open_end*: should the heads be open, if arrows are drawn, and if *nheads* is -1 then this is the *linewidth* to use when drawing the X
  - *nheads_start* (optional) : how many heads to draw at the start
  - *open_start* (optional) : should the heads be open at the start.
- *circle*: draw a circle. The params are:
  - *pixels*: the size of the circle
  - *open*: if 0 the circle is filled, if 1 it is wireframe
- *triangle*: draw a triangle (like a constraint). The params are:
  - *pixels*: the size of the triangle
  - *open*: if 0 the triangle is filled, if 1 it is wireframe
  - *tail*: draw a tail on it
- *helix*: draw a spring-like helix. The params are:
  - *ntwists*: the number of twists
  - *radius*: the radius as a factor of the length

ic_boco_apply_attr_symbol *group bctype setname icontype parsed params labels vis lvis*

Smarter wrapper around the below functions.

ic_boco_set_attr_symbol_group_visible *group bctype vis lvis*

Easier way to set visibility by group and bctype

ic_boco_add_attr_symbol *bctype group icontype set params label vis lvis*

The new version which can pull out distributed values from nodes and elements. *type* is the *bctype* this corresponds to. *group* is the group or part this attaches to. *icontype* is arrow, helix, etc. *set* controls how new symbols replace old ones. *params* is a list of {name value expr} tuples. *expr* is a list of bcfield names that get multiplied together. *label* is a list of *text*, *paramname*, *text*, *paramname* .. that gets concatenated. *vis* is a flag saying whether it should be initially visible or not. *lvis* is 1 if the label should be drawn once, 0 not, 2 on all points Return value is a symbol id that can be used in the functions below.

ic_boco_delete_attr_symbol *id*

Deletes a symbol.

ic_boco_list_attr_symbols *bctype* [""] *group* [""] *set* [""]

Lists symbols.

ic_boco_modify_attr_symbol_params *id params*

Modifies params.

ic_boco_modify_attr_symbol_label *id label*

Modifies labels.

ic_boco_modify_attr_symbol_visible *id vis lvis*

Modifies visibility.

ic_boco_get_attr_symbol_visible *id*

Gets visibility flags - icon and label.

ic_boco_create_trivial *bocofilename* [./family_boco_trivial]

Creates a trivial (no boundary conditions) *family_boco* file with filename *bocofilename* for the loaded unstruct mesh. This function returns the filename created.

ic_boco_ensure_part_exists *nfam*

Makes sure a part name exists.

ic_boco_set_for_objects *objects bocos pre nfam*

Applies a list of BCs to either a set of geometrical entities or a mesh subset. *objects* is a list of type, names, ... The *types* can be surface, curve, point, material, subset, or unsmap. Families are composed of "*groups*" separated by commas. Each group is also a family whether it's used or not. Groups are named G0, G1, etc, where the prefix G could be anything such as CN for constraints. This function returns a list of the new family names that have been created.

Note:

If you pass a *unsmap* in as one of the objects parameters, this function will delete it. The *nfam* argument is the name of a group to use, instead of checking to see if a new one should be created

ic_boco_apply_generic_icon *fam obtype ctype params vis only_dims* [""]

Applies a non-solver specific icon to a particular family.

ic_boco_apply_solver_icon *fam bctype icontype iconparams vis only_dims* [""] *bcargs* [""]

Applies a solver specific icon to a particular family.

ic_boco_apply_generic_icon_to_group *group ctype params vis only_dims* [""]

Applies a non-solver specific icon to a particular family.

ic_boco_apply_solver_icon_to_group *group bctype icontype iconparams vis bcargs* [""]

Applies a solver specific icon to a particular group.

ic_boco_apply_solver_icon_to_part *part bctype icontype iconparams vis bcargs* [""]

Applies a solver specific icon to a particular part.

unused__ic_boco_set_visible *bcargs bctypes on*

Sets the icon visibility. *bcargs* are something like FORCE_NAME F1

ic_boco_delete_icons_for_bctypes *fam bctypes*

Removes the icons for a particular family/bctype combination.

ic_boco_set_family_visible *fams bctypes on*

Sets the icon visibility by family. If *fams* is empty then this implies all families.

ic_boco_remove_objects_from_groups *objlist bcargs*

Removes the given objects from all groups that have a BC with a specified name.

ic_boco_add_parts_and_subsets_to_group *name parts_or_subsets*

Adds the parts and subsets to the APPLY_TO_PARTS and APPLY_TO_SUBSETS pseudo-BC's on the named group.

ic_boco_what_groups_go_with_part_or_subset *type name*

This routine is called when objects are moved into and out of parts and subsets.

ic_boco_change_part *oldfam newpart*

Utility function to move stuff from one part to another.

ic_boco_change_subset *oldfam subset add*

For a given family, if you move something in that family into or out of a subset, this returns the new family.

ic_boco_add_or_remove_bc_icons *on_fam add objects*

For every icon on family *oldfam*, remove the locations of the objects. For every icon on *newfam*, add the locations.

ic_boco_replace_arg *fams replace_types replace_pos replace_val*

Replaces a certain argument in the BCs for a given set of families with a different value.

ic_boco_delete_group_with_bcs *bcargs*

Removes everything from groups that have a specific *bctype* and set of arguments, and delete those groups. Note that this will never get called with part groups.

ic_boco_list_icons *fam* [""] *type* [""]

Lists all the BC icons.

ic_boco_delete_group *group*

Removes everything from groups and delete those groups.

ic_boco_clear_family *fam*

Clears the BCs and icons from one family.

ic_boco_set_part_color *famname update_uns* [1]

Looks for a group inside this family that is a part name, and define the color of the family based on the value of the name. Note that it takes only the last component of the part name so that things do not shift around when you put them in assemblies.

ic_boco_nastran_csystem *what args*

Sets Nastran-type coordinate systems.

ic_boco_delete_unused *keep_groups*

Cleans up the set of BCs a bit. Keeps the singleton groups if *keep_groups* is 1.

ic_boco_list_unused

Lists the families that have no mesh or geometry in them.

ic_boco_get_fams_of_part *partname*

Returns the list of families belonging to the given *partname*.

ic_boco_get_part_of_fam *famname*

Returns the part to which the given family *famname* belongs.

ic_solver_mesh_info *solver*

Utility Functions

ic_mess *args*

Prints the given message to the message window or the standard output if MED is being run in batch mode. There may be 1 or 2 arguments. The first is the string and the second if given is the color to print the message in.

ic_print_error_log *res*

Prints the error log **res** to the screen in red. If there are more than 64 lines, they will all be saved to a temporary file such as the default /tmp/ERROR_LOG0.tmp

ic_write_file *name text*

Creates a file with some stuff in it.

ic_quit

Quits MED and save the current project settings.

ic_dialog *args*

Calls the **tk_dialog** utility function.

ic_question *args*

Queries you to input a string value.

ic_confirm *mess*

Presents a dialog with a message and one button that says OK that just dismisses the dialog.

ic_yes_or_no *mess*

Presents a dialog with a message and 2 buttons, which say Yes or No. Returns 1 or 0 depending on which is pressed.

ic_multiple_choice *mess args*

Presents a dialog with a message and multiple buttons with the given labels. Returns the ordinal of the one that is pressed.

ic_multiple_choice_default *default mess args*

Same as **ic_multiple_choice** but *default* gives the index of the button that is to be the default button for the dialog. If less than zero then there will not be any default button.

ic_pause *ms*

Waits for *ms* milliseconds, process regular Tk events, and then returns.

ic_batch_mode

Checks for batch mode.

ic_run_application_command *dir progdir progname arguments*

Runs some external application.

ic_run_application_batch *dir progdir progname envname arguments logfile* [""]

Runs some external application in batch mode

ic_run_application_exec *dir progdir progname arguments*

Runs some external application, given the full path.

ic_exec *args*

A simplified version of this.

ic_run_application_direct *dir progdir progname envname arguments*

Runs an application.

ic_remove_duplicate *names*

Removes duplicates from list of *names*.

ic_undo

Undoes the previous action.

ic_redo

Redoes the previous undone action.

ic_undo_group_begin *text* [undo_group]

Sets undo group begin.

ic_undo_group_end *text* [undo_group]

Sets undo group end.

ic_undo_suspend

Suspends undo logging.

ic_undo_suspended

State of undo manager.

ic_undo_resume

Resumes undo logging.

ic_undo_start

Starts undo handler; started by default.

ic_undo_stop

Stops undo handler; removes undo log.

ic_undo_clear

Clears undo events and restarts undo log.

ic_archive_dir *dir archive*

Allows you to tar and gzip the directory *dir* as *archive*.

ic_unarchive_file *file dest*

Allows you to untar and gunzip the archive *file* in directory *dest*.

ic_stopwatch_start

Starts a stopwatch, in milliseconds.

ic_stopwatch_get_elapsed *stop_the_watch* [0]

Gets the elapsed time, in milliseconds, since start of the stopwatch. Optional argument *stop_the_watch* will also stop the watch.

ic_stopwatch_end

Stops the current stopwatch.

ic_list_uniquify *list*

Uniquifies given TCL list *list*. Note that this function returns the list sorted.

ic_list_remove_duplicated *list*

For every non-unique element in given TCL list *list*, remove all instances. Note that this function returns the list sorted.

ic_list_get_duplicated *list*

This returns the duplicated items in a list

ic_list_remove_from_grouped *groups orig*

Takes a list of groups and an original list and returns items in *orig* that are not in groups

ic_list_median *list*

Gets the median in the list.

ic_chdir *dir*

Changes working directory.

ic_check_licensing_aienv

Checks to see if ai*env licensing is enabled.

ic_reinit_geom_objects *all* [""]

Redefines graphics for geometry.

ic_error

For testing.

IC3M Functions

ic_rmi_ic3m_batch *args*

Runs **IC3M** in batch mode. The options are:

- **file** : the path to the settings file
- **geom** : 1 if the geometry for intake/exhaust valves should be created, 0 if not (default 1)
- **mesh** : 1 if the mesh for intake/exhaust valves should be created, 0 if not (default 1)
- **comb** : 1 if geometry/mesh for combustion dome should be created, 0 if not (default 1)
- **piston** : 1 if geometry/mesh for piston bowl should be created, 0 if not (default 1)
- **output_to** : *starcd* or *fluent*, if output mesh should be created (default none)
- **output** : transient or steady_state, if output mesh to STAR-CD or Fluent should be created (default none)

Optional options: (overwrites the values from the settings file)

- **platform** : output platform for the STAR-CD run
- Possible values for **platform** : (default is value from the settings file)
  - **sgi_m2**
  - **sgi64_m4**
  - **aix_generic**
  - **sun_generic**
  - **osf1**
  - **hp_generic**

- **linux**

- **precision** : precision for that platform
- Possible values for **precision** : (default is value from the settings file)
  - **1**
  - **2**
- Only used if output mesh is set to steady_state : (defaults are values from the settings file)
  - **angle_dca** : angle DCA
  - **use_cyl_len** : if 1 use also cylinder length and number of cylinder layers:
    - → **cyl_length** : cylinder length
    - → **nlayers_cyl** : number of cylinder layers

Example usage: **ic_rmi_ic3m_batch***file* engine/test.def *output_to* starcd *output* transient

XML Report Editing and Modification Functions

ic_reports_write_mesh_report_ *what file args*

Write a *mesh_report* XML file of the given subset name *what* to the given *file*. If {"FILE" == $what}, then you must specify "-domain $domainfilename" and it will write a report for that entire mesh. Arguments can be: -write_summary $val, -write_quality $val, -write_diagnostics $val, -format (xml|html)

Usage: **ic_reports_write_mesh_report** All test.xml -format xml

Usage: **ic_reports_write_mesh_report** FILE test.html -format html -domain project1.uns

ic_reports_write_mesh_report *what file args*

ic_reports_compare_mesh *stdmesh curmesh* [""]

ic_reports_write_assembly_report *infiles informat outfile outformat args*

Regression Testing Functions

ic_regression_list *args*

General resultlib read function. Reads the files specified by the file list using the reader specified by the backend. If create is one, the unstructured/structured meshes will be recreated.

ic_regression_test *action cfile match* [-fuzzy] *mesh* [-info] *compareopts* [""] *prec* [double] *tstep* [0] *dir* [""] *cstyle* [domain] *factory* [""] *meshimpl* [""]

Carry out regression test.

- *action* : create or run

- *cfile* : is the comparison file name. If the macro $D is in the name, the directory given by the *dir* parameter will be substituted for it. All of the rest of the arguments are optional.
- *match* : {-exact | -fuzzy} If -exact, then the all portions of the mesh comparison must match exactly. If -fuzzy, then the comparison does not have to be exact (ie, allow discrepancies up to certain percentage of the checks).
- *mesh* : {-mapped | -info | -nofamily | -nofamilycounts | -grid | -elements | -all}* this list tells what the regression test will match. If -mapped is specified, then the nodes of the mesh are mapped to the comparison mesh. If -info in list, the counts of nodes, elements, structured domains, families and family members will all be checked. If -nofamily is specified with -info, the family name portion of the mesh will not be checked. If no family counts is specified, the family names will be checked, but no match in counts of the family members will be required. If -grid is specified, then the nodes of the mesh will be checked. If -elements is specified, then the finite elements will be checked. If -nofamily (or -nofamilycounts) specified with -elements, then the family id will not be checked. If -all is specified, then information, grid and elements will all be checked. The -nofamily and -nofamilycounts will still act as qualifiers in this case.
- *compareopts* : any additional comparison options. The match and grid arguments set the basic comparison options for the regression test. You can augment the comparison (for instance changing the tolerances for distances) by setting this option. These options will be appended to those determined by the match and mesh options.
- *prec*: {double | float} precision of grid coordinates. Default is double.
- *tstep* : timestep to load. Reserved for future use.
- *dir* : is the directory the comparison file can be found. Default is env(REGRESSION_DIR) if set, or the current working directory if not. Only used if $D is in the comparison file name.
- *cstyle* : is the style of the comparison file. The only options are script or domain; CGNS is reserved for future use.
- *factory* : the factory for the backend for the comparison file. If not given, it will use the domain factory which has an internal domain file reader (only relevant in case of the comparison file style not being a script file).
- *meshimpl* : the mesh implementation. If not given, this will use med_impl.

ic_regression_test_rfint *rfint action cfile match* [-fuzzy] *mesh* [-info] *compareopts* [""] *prec* [double] *tstep* [0] *dir* [""] *cstyle* [domain] *factory* [""] *meshimpl* [""]

The only difference between this and **ic_regression_test** is that it allows the mesh to be computed externally. For instance, load mesh tests would use this interface as opposed to the regression_test interface.

Workbench Integration

ic_wb2 *file* [""]

Updates Workbench 2.0 upstream geometry

file	Geometry or external CAD file

ic_wb2_set_parameter *type param value invoke* [0]

Sets a parameter in the Workbench "Parameter Set" gui.

Usage: **ic_wb2_set_parameter** TYPE PARAMETER_NAME VALUE

TYPE: input or output or user_defined. PARAMETER_NAME: parameter label/name, VALUE: value of PARAMETER_NAME.

ic_wb2_set_icemcfd_parameter *param value*

Copies a user value of a Workbench input "Parameter Set" entity to the ICEM CFD parameter.

Usage: **ic_wb2_set_icemcfd_parameter** PARAMETER_NAME VALUE.

Note: PARAMETER_NAME must be defined before using **ic_wb2_set_icemcfd_parameter**.

ic_wb2_get_parameter *type param*

Returns the value of a Workbench "Parameter Set" entity.

Usage: set value [**ic_wb2_get_parameter** TYPE PARAMETER_NAME]

ic_wb2_delete_parameter *param invoke* [0]

Deletes a Workbench entity from the "Parameter Set" entity.

Usage: **ic_wb2_delete_parameter** PARAMETER_NAME.

ic_wb2_update_parameters *types* [input output] *invoke* [0]

Refreshes all current input/output "Parameter Set" entities at once, or choose only `input` or `output` using the TYPE switch.

Usage: **ic_wb2_update_parameters** [TYPE].

ic_wb2_delete_parameters_by_pattern *type pattern*

Deletes Workbench "Parameter Set" entities that meet a specified criterion.

Usage: **ic_wb2_delete_parameters_by_pattern** TYPE PATTERN

For example, ic_wb2_delete_parameters_by_pattern TYPE "Number of * elements"

For example, ic_wb2_delete_parameters_by_pattern TYPE "Min quality of *"

ic_wb2_get_parameters *type*

Returns all pairs of PARAMETER_NAME VALUE of type TYPE.

Usage: set values [**ic_wb2_get_parameters** TYPE]