

CFD EXPERTS

Simulate the Future

WWW.CFDEXPERTS.NET



©2021 ANSYS, Inc.

All Rights Reserved.

Unauthorized use, distribution
or duplication is prohibited.

Ansys TurboGrid Reference 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

ANSYS, Inc. and Ansys Europe, Ltd. are UL registered ISO 9001: 2015 companies.

Copyright and Trademark Information

© 2021 ANSYS, Inc. Unauthorized use, distribution or duplication is prohibited.

Ansys, Ansys Workbench, AUTODYN, CFX, FLUENT and any and all ANSYS, Inc. brand, product, service and feature names, logos and slogans are registered trademarks or trademarks of ANSYS, Inc. or its subsidiaries located in the United States or other countries. ICEM CFD is a trademark used by ANSYS, Inc. under license. CFX is a trademark of Sony Corporation in Japan. All other brand, product, service and feature names or trademarks are the property of their respective owners. FLEXlm and FLEXnet are trademarks of Flexera Software LLC.

Disclaimer Notice

THIS ANSYS SOFTWARE PRODUCT AND PROGRAM DOCUMENTATION INCLUDE TRADE SECRETS AND ARE CONFIDENTIAL AND PROPRIETARY PRODUCTS OF ANSYS, INC., ITS SUBSIDIARIES, OR LICENSORS. The software products and documentation are furnished by ANSYS, Inc., its subsidiaries, or affiliates under a software license agreement that contains provisions concerning non-disclosure, copying, length and nature of use, compliance with exporting laws, warranties, disclaimers, limitations of liability, and remedies, and other provisions. The software products and documentation may be used, disclosed, transferred, or copied only in accordance with the terms and conditions of that software license agreement.

ANSYS, Inc. and Ansys Europe, Ltd. are UL registered ISO 9001: 2015 companies.

U.S. Government Rights

For U.S. Government users, except as specifically granted by the ANSYS, Inc. software license agreement, the use, duplication, or disclosure by the United States Government is subject to restrictions stated in the ANSYS, Inc. software license agreement and FAR 12.212 (for non-DOD licenses).

Third-Party Software

See the [legal information](#) in the product help files for the complete Legal Notice for Ansys proprietary software and third-party software. If you are unable to access the Legal Notice, contact ANSYS, Inc.

Published in the U.S.A.

Table of Contents

1. Ansys TurboGrid Launcher	9
1.1. The Ansys TurboGrid Launcher Interface	9
1.1.1. Menu Bar	9
1.1.1.1. File Menu	10
1.1.1.1.1. Save As	10
1.1.1.1.2. Quit	10
1.1.1.2. Edit Menu	10
1.1.1.2.1. Clear	10
1.1.1.2.2. Find	10
1.1.1.2.3. Options	10
1.1.1.2.3.1. Graphical User Interface Style	10
1.1.1.2.3.2. Font and Formatted Font	10
1.1.1.3. CFX Menu	10
1.1.1.3.1. TurboGrid 2021 R2	10
1.1.1.3.2. Other Ansys CFX Applications	11
1.1.1.4. Show Menu	11
1.1.1.4.1. Installation	11
1.1.1.4.2. All	11
1.1.1.4.3. System	11
1.1.1.4.4. Variables	11
1.1.1.5. Tools Menu	11
1.1.1.5.1. Ansys Client Licensing Utility	11
1.1.1.5.2. Command Line	11
1.1.1.5.3. Edit File	12
1.1.1.6. User Menu	12
1.1.1.7. Help Menu	12
1.1.2. Toolbar	12
1.1.3. Working Directory Selector	13
1.1.4. Output Window	13
1.2. Customizing the Ansys TurboGrid Launcher	13
1.2.1. CCL Structure	13
1.2.1.1. GROUP	14
1.2.1.2. APPLICATION	14
1.2.1.2.1. Including Environment Variables	16
1.2.1.3. DIVIDER	16
1.2.2. Example: Adding the Windows Calculator	17
2. CFX Command Language	19
2.1. Introduction	19
2.2. CCL Syntax	19
2.2.1. Basic Terminology	20
2.2.1.1. The Data Hierarchy	20
2.2.2. Simple Syntax Details	20
2.2.2.1. Case Sensitivity	20
2.2.2.2. CCL Names Definition	21
2.2.2.3. Indentation	21
2.2.2.4. End of Line Comment Character	21
2.2.2.5. Continuation Character	21
2.2.2.6. Named Objects	22
2.2.2.7. Singleton Objects	22

2.2.2.8. Parameters	22
2.2.2.9. Lists	22
2.2.2.10. Parameter values	22
2.2.2.10.1. String	22
2.2.2.10.2. String List	23
2.2.2.10.3. Integer	23
2.2.2.10.4. Integer List	23
2.2.2.10.5. Real	23
2.2.2.10.6. Real List	24
2.2.2.10.7. Logical	24
2.2.2.10.8. Logical List	24
2.2.2.11. Escape Character	24
2.3. Object Creation and Deletion	24
2.4. Ansys CFX Expression Language	25
2.4.1. CEL Functions, Constants and System Variables	26
2.4.1.1. CEL Standard Functions	26
2.4.1.2. CEL Constants	26
3. Command Actions	27
3.1. Introduction	27
3.1.1. Command Actions Example	27
3.2. File Operations from the Command Editor Dialog Box	28
3.2.1. Save State Files	28
3.2.1.1. savestate Command Examples	29
3.2.2. Read State Files	30
3.2.2.1. readstate Command Examples	31
3.2.3. Save Topology Files	32
3.2.3.1. savetopology Command Example	33
3.2.4. Save Mesh Files	33
3.2.4.1. savemesh Command Examples	33
3.2.5. Save Blade Files	33
3.2.5.1. saveblade Command Example	34
3.2.6. Create Session Files	34
3.2.6.1. session Command Examples	35
3.2.7. Read Session Files	36
3.2.7.1. readsession Command Examples	36
3.2.8. Create Hardcopy	37
3.2.9. Export Geometry	37
3.2.9.1. tetin Command Example	37
3.3. Quantitative Calculations in the Command Editor Dialog Box	37
3.3.1. Function Calculation	38
3.3.1.1. Expression Specification	39
3.3.1.2. Axis Specification	39
3.3.1.3. Quantitative Function List	39
3.3.1.3.1. area	39
3.3.1.3.2. areaAve	40
3.3.1.3.3. areaInt	40
3.3.1.3.4. ave	40
3.3.1.3.5. count	40
3.3.1.3.6. length	40
3.3.1.3.7. lengthAve	40
3.3.1.3.8. lengthInt	41

3.3.1.3.9. maxVal	41
3.3.1.3.10. minVal	41
3.3.1.3.11. probe	41
3.3.1.3.12. sum	41
3.3.1.3.13. volume	41
3.3.1.3.14. volumeAve	41
3.3.1.3.15. volumeInt	41
3.4. Other Commands	42
3.4.1. Deleting Objects	42
3.4.2. Viewing a Chart	42
3.4.3. Creating a Mesh	42
4. Power Syntax	43
4.1. Introduction	43
4.2. Examples of Power Syntax	44
4.2.1. Example 1: Using a for Loop	44
4.2.2. Example 2: Creating a Simple Subroutine	44
4.3. Predefined Power Syntax Subroutines	45
4.3.1. evaluate(Expression)	45
4.3.2. getValue(Object Name, Parameter Name)	45
4.3.3. showPkgs()	45
4.3.4. showSubs()	46
4.3.5. showVars()	46
4.3.6. verboseOn()	46
5. Line Interface Mode	47
5.1. Introduction	47
5.2. Line Interface Mode	48
5.2.1. Lists of Commands	49
5.2.2. Viewer Hotkeys	49
5.2.3. Calculator	49
5.2.4. getstate Command	49
5.2.5. Repeating CCL Commands	49
5.2.6. Executing a Shell Command	49
5.2.7. Quitting	49
5.2.8. Example	50
5.3. Batch Mode	50
5.3.1. Example: Generating a Similar Mesh from Different Curve Files	50
6. Meshing Reference	53

List of Figures

1.1. Ansys TurboGrid Launcher 9

Chapter 1: Ansys TurboGrid Launcher

This chapter describes the Ansys TurboGrid Launcher in detail:

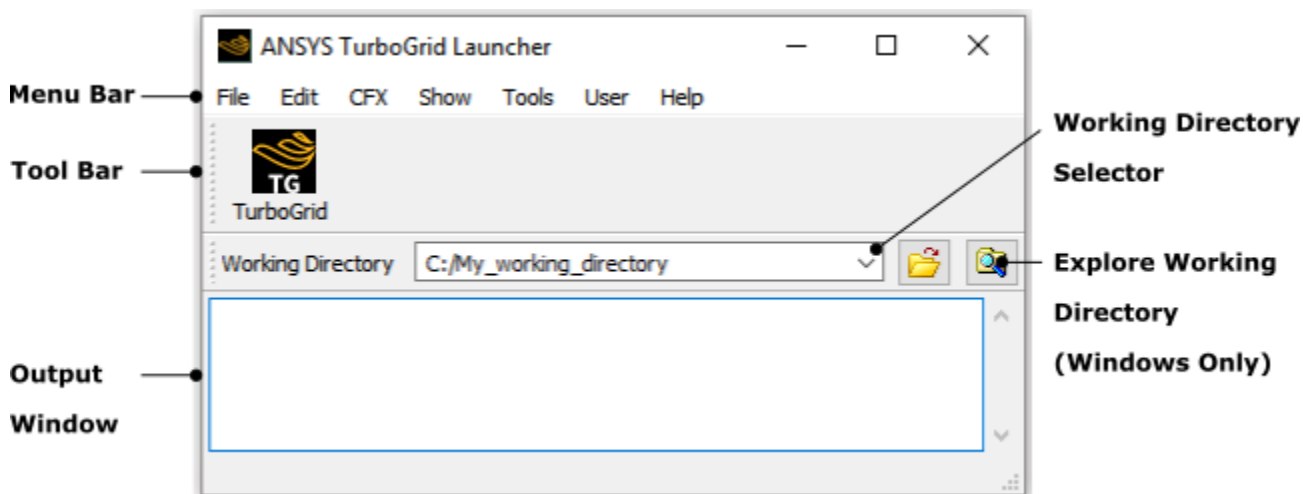
1.1. The Ansys TurboGrid Launcher Interface

1.2. Customizing the Ansys TurboGrid Launcher

1.1. The Ansys TurboGrid Launcher Interface

The layout of the Ansys TurboGrid Launcher is shown below:

Figure 1.1: Ansys TurboGrid Launcher



The Ansys TurboGrid Launcher consists of a menu bar, a toolbar for launching applications, a working directory selector, and an output window where messages are displayed. On Windows platforms, an icon to start Windows Explorer in the working directory appears next to the directory selector.

The following sections describe parts of the launcher:

1.1.1. Menu Bar

1.1.2. Toolbar

1.1.3. Working Directory Selector

1.1.4. Output Window

1.1.1. Menu Bar

The Ansys TurboGrid Launcher menus are described briefly in the following table, and in more detail following the table.

1.1.1.1. File Menu

Saves the contents of the text output window and to close the Ansys TurboGrid Launcher.

1.1.1.1.1. Save As

Saves the contents of the output window to a file.

1.1.1.1.2. Quit

Shuts down the Ansys TurboGrid Launcher. Any programs already launched will continue to run.

1.1.1.2. Edit Menu

Clears the text output window, finds text in the text output window and sets options for the Ansys TurboGrid Launcher.

1.1.1.2.1. Clear

Clears the output window.

1.1.1.2.2. Find

Displays a dialog box where you can search the text in the output window.

1.1.1.2.3. Options

Presents the **Options** dialog box, which enables you to change the appearance of the Ansys TurboGrid Launcher. Once you have configured the settings, click **Apply** to apply the settings tentatively. Click **OK** to accept the settings. Click **Restore** to revert the settings to the previously accepted configuration.

1.1.1.2.3.1. Graphical User Interface Style

You can choose any one of the listed interface styles to change the look and feel of the user interface.

1.1.1.2.3.2. Font and Formatted Font

The button to the right of **Font** sets the font used anywhere outside the text output window. The button to the right of **Formatted Font** applies only to the text output window. Clicking either of these buttons opens the **Select Font** dialog box.

1.1.1.3. CFX Menu

1.1.1.3.1. TurboGrid 2021 R2

Runs Ansys TurboGrid, with the working directory as specified in [Working Directory Selector \(p. 13\)](#).

1.1.1.3.2. Other Ansys CFX Applications

The Ansys TurboGrid Launcher will search for installed Ansys CFX applications (for example, CFX-Pre, CFD-Post) and provide a menu entry to launch each application. If an application is not found, you can add it; for details, see [Customizing the Ansys TurboGrid Launcher \(p. 13\)](#).

1.1.1.4. Show Menu

1.1.1.4.1. Installation

Displays information about the version of Ansys TurboGrid that you are running.

1.1.1.4.2. All

Displays all of the available information, including information about your system, installation and variables.

1.1.1.4.3. System

Displays information about the Ansys TurboGrid installation and the system on which it is being run.

1.1.1.4.4. Variables

Displays the values of all the environment variables that are used in Ansys TurboGrid.

1.1.1.5. Tools Menu

Enables you to access license-management tools and a command line for running other Ansys CFX utilities.

1.1.1.5.1. Ansys Client Licensing Utility

Enables you to configure connections to Ansys License Managers.

1.1.1.5.2. Command Line

Starts a command window from which you can run any of the Ansys TurboGrid commands via the command line interface. The command line will be set up to run the correct version of Ansys TurboGrid and the commands will be run in the current working directory.

If you do not use the **Tools > Command Line** command to open a command window, then you will have to either type the full path of the executable in each command, or explicitly set your operating system path to include the `<CFXROOT>/bin` directory.

You may want to start Ansys TurboGrid from the command line rather than by clicking the appropriate button on the Ansys TurboGrid Launcher for the following reasons:

- Ansys TurboGrid contains some utilities (for example, a parameter editor) that can be run only from the command line.

- You may want to specify certain command line arguments when starting up a component so that it starts up in a particular configuration.
- If you are having problems with a component, you may be able to get a more detailed error message by starting the component from the command line than you would get if you started the component from the launcher. If you start a component from the command line, any error messages produced are written to the command line window.

1.1.1.5.3. Edit File

Opens a browser to edit the text file of your choice in a platform-native text editor. Which text editor is called is controlled by the settings in `<CFXROOT>/etc/launcher/shared.ccl`.

1.1.1.6. User Menu

The **User** menu is provided as an example. You can add your own applications to this menu, or create new menus. For details, see [Customizing the Ansys TurboGrid Launcher \(p. 13\)](#).

1.1.1.7. Help Menu

The **Help** menu has the following commands:

TurboGrid Launcher

Opens [Using the Ansys TurboGrid Launcher in the TurboGrid Introduction](#).

Contents

Opens a page that lists various help resources associated with this product.

[various help resources]

Each of these commands goes directly to a particular help resource.

Ansys Product Improvement Program

Provides a brief description of, and enables you to control participation in, the Ansys Product Improvement Program.

About TurboGrid Launcher

This gives the point releases and software patches that are installed.

Help on Help

Opens documentation about the help system: [Ansys TurboGrid Help and Conventions in the TurboGrid Introduction](#).



1.1.2. Toolbar

The toolbar contains shortcuts to the main components of Ansys CFX, for example Ansys TurboGrid, CFX-Pre, CFX-Solver Manager and CFD-Post. Pressing any of the buttons will start up the component

in the specified working directory. The equivalent menu entries for launching the components also show a keyboard shortcut that can be used to launch the component.

1.1.3. Working Directory Selector

While running Ansys TurboGrid, all the files that are created will be stored in the working directory. To change the working directory, you can do any of the following:

- Type the directory name into the box and press **Enter**.
- Click the down-arrow icon () next to the directory name. This displays a list of recently used directories.
- Click *Browse*  to browse to the directory that you want.

1.1.4. Output Window

The output window is used to display information from commands in the **Show** menu. You can right-click in the output window to show a shortcut menu with the following options:

- **Find**: Displays a dialog box where you can enter text to search for in the output.
- **Select All**: Selects all the text.
- **Copy Selection**: Copies the selected text.
- **Save As**: Saves the output to a file.
- **Clear**: Clears the output window.

1.2. Customizing the Ansys TurboGrid Launcher

Many parts of the Ansys TurboGrid Launcher are driven by CCL commands contained in configuration files. Some parts of the launcher are not editable (such as the **File**, **Edit** and **Help** menus), but others parts enable you to edit existing actions and create new ones (for example, launching your own application from the **User** menu). The following sections outline the steps required to configure the launcher. The configuration files are located in the `<CFXROOT>/etc/launcher/` directory (where `<CFXROOT>` is the path to your installation of Ansys TurboGrid). You can open these files in any text editor, but you should not edit any of the configuration files provided by Ansys TurboGrid, other than the `User.ccl` configuration file.

1.2.1. CCL Structure

The configuration files contain CCL objects that control the appearance and behavior of menus and buttons that appear in the Ansys TurboGrid Launcher. There are three types of CCL objects: `GROUP`, `APPLICATION` and `DIVIDER` objects. The fact that there are multiple configuration files is not important; applications in one file can refer to groups in other files.

An example of how to add a menu item for the Windows calculator to the launcher is given in [Example: Adding the Windows Calculator \(p. 17\)](#).

1.2.1.1. GROUP

GROUP objects represent menus and toolbar groups in the Ansys TurboGrid Launcher. Each new GROUP creates a new menu and toolbar. Nothing will appear in the menu or toolbar until you add APPLICATION or DIVIDER objects to the group. An example of a GROUP object is given below:

```
GROUP: CFX
  Position = 200
  Menu Name = &CFX
  Show In Toolbar = Yes
  Show In Menu = Yes
  Enabled = Yes
END
```

- The group name is set after the colon. In this case, it is "CFX". This is the name that APPLICATION and DIVIDER objects will refer to when you want to add them to this group. This name should be different to all other GROUP objects.
- Position refers to the position of the menu relative to others. The value should be an integer between 1 and 1000. Groups with a higher Position value, relative to other groups, will have their menu appear further to the right in the menu bar. Referring to [Figure 1.1: Ansys TurboGrid Launcher \(p. 9\)](#), CFX has a lower position value than the Ansys group. The **File** and **Edit** menus are always the first two menus and the **Help** menu is always the last menu.
- The title of the menu is set under Menu Name (this menu has the title **CFX**). The optional ampersand is placed before the letter that you want to act as a menu accelerator (for example, **Alt+C** displays the **CFX** menu). You must be careful not to use an existing menu accelerator.
- The creation of the menu or toolbar can be toggled by setting the Show in Menu and Show in Toolbar options to Yes or No respectively. For example, you may want to create a menu item but not an associated toolbar icon.
- Enabled sets whether the menu/toolbar is available for selection or is grayed out. Set the option to No to gray it out.

1.2.1.2. APPLICATION

APPLICATION objects create entries in the menus and toolbars that will launch an application or run a process. Two examples are given below with an explanation for each parameter. The first example creates a menu entry in the **Tools** menu that opens a command line window. The second example creates a menu entry and toolbar button to start CFX-Solver Manager.

```
APPLICATION: Command Line 1
  Position = 300
  Group = Tools
  Tool Tip = Start a window in which CFX commands can be run
  Menu Item Name = Command Line
  Command = <windir>\system32\cmd.exe
  Arguments = /c start
  Show In Toolbar = No
  Show In Menu = Yes
  Enabled = Yes
  OS List = winnt
END
APPLICATION: CFXSM
  Position = 300
  Group = CFX
```

```

Tool Tip = Launches Ansys CFX-Solver Manager
Menu Item Name = CFX-Solver Manager
Command = cfx5solve
Show In Toolbar = Yes
Show In Menu = Yes
Enabled = Yes
Toolbar Name = Ansys CFX-Solver Manager
Icon = LaunchSolveIcon.xpm
Shortcut = Ctrl+S
END

```

- The application name is set after the colon, in the first example it is "Command Line 1". This name should be different to all other APPLICATION objects.
- **Position:** sets the relative position of the menu entry. The value should be an integer between 1 and 1000. The higher the value, relative to other applications that have the same group, the further down the menu or the further to the right in a toolbar the entry will appear. If you do not specify a position, the object assumes a high position value (so it will appear at the bottom of a menu or at the right of a group of buttons).
- **Group:** sets the GROUP object to which this application belongs. The value must correspond to the name that appears after "GROUP:" in an existing GROUP object. The menu and/or toolbar entry will not be created if you do not specify a valid group name. The GROUP object does not have to be in the same configuration file.
- **Tool Tip:** displays a message when the mouse pointer is held over a toolbar button. In the 'Command Line 1' example above, the Tool Tip entry is not used since a toolbar button is not created. This parameter is optional.
- **Menu Item Name:** sets the name of the entry that will appear in the menu. If you do not specify a name, the name is set to the name of the APPLICATION: object. The optional ampersand is placed before the letter that you want to have act as a menu accelerator (for example, **alt+c** then **s** will start CFX-Solver Manager. **Alt+c** selects the **CFX** menu and "s" selects the entry from the menu). You must be careful not to use an existing menu accelerator.
- **Command:** contains the command to run the application. The path can be absolute (that is, use a forward slash to begin the path on Linux, or a drive letter on Windows). If an absolute path is not specified, a relative path from `<CFXROOT>/bin/` is assumed. If no command is specified, the menu item/toolbar button will not appear in the Ansys TurboGrid Launcher. The path and command are checked when the launcher is started. If the path or command does not exist, the menu item/toolbar button will not appear in the launcher. You may find it useful to include environment variables in a command path; for details, see [Including Environment Variables \(p. 16\)](#).
- **Arguments:** specifies any arguments that need to be passed to the application. The arguments are appended to the value you entered for Command. You do not need to include this parameter as there are no arguments to pass. You may find it useful to include environment variables in the arguments; for details, see [Including Environment Variables \(p. 16\)](#).

Distinct arguments are space-separated. If you need to pass an argument that contains spaces (for example, a Windows filepath) you should include that argument in double quotes, for example:

```
Arguments = "C:\Documents and Settings\User" arg2 arg3
```

- **Show In Toolbar:** determines if a toolbar button is created for the application. This optional parameter has a default value of **Yes**.

- **Show In Menu:** determines if a menu entry is created for the application. This optional parameter has a default value of **Yes**.
- **Enabled:** allows you to gray out the menu entry and toolbar button. Set this parameter to **No** to gray out the application. This optional parameter has a default value of **Yes**.
- **OS List** is an optional parameter that allows you to set which operating system the application is suitable for. If **OS List** is not supplied, the launcher will attempt to create the menu item and toolbar button on all platforms.

For example, the command to open a command line window varies depending on the operating system. In the 'Command Line 1' example above, the application only applies to Windows platforms. To complete the OS coverage, the launcher configuration files contain more 'Command Line' applications that apply to different operating systems.

OS List can contain the following values: **winnt** (Windows), **linux-ia64** (64-bit Linux).

- **Toolbar Name:** sets the name that appears on the toolbar button. This parameter is optional (since you may only want to show an icon).
- **Icon:** specifies the icon to use on the toolbar button and in the menu item. The path can be absolute (that is, use a forward slash to begin the path on Linux, or a drive letter on Windows). If an absolute path is not specified, a relative path from **<CFXROOT>/etc/icons** is assumed. The following file formats are supported for icon image files: Portable Network Graphics (**png**), Pixel Maps (**ppm**, **xpm**) and Bitmaps (**bmp**). Other icons used in the launcher are 32 pixels wide and 30 pixels high. This parameter is optional. If it is not included, an icon will not appear.
- **Shortcut:** specifies the keyboard shortcut that can be pressed to launch the application. You must be careful not to use a keyboard shortcut that is used by any other **APPLICATION** object.

1.2.1.2.1. Including Environment Variables

It can be useful to use environment variables in the values for some parameters. You can specify an environment variable value in any parameter by including its name between the **< >** symbols. In the 'Command Line 1' example above, **<windir>** is used in the **Command** parameter so that the command would work on different versions of Windows. **<windir>** is replaced with the value held by the **windir** environment variable. The **Command** and **Argument** parameters are the only parameters that are likely to benefit from using environment variables. Environment variables included in the **Arguments** parameter are expanded before they are passed to the application.

1.2.1.3. DIVIDER

DIVIDER objects create a divider in a menu and/or toolbar (see the **Tools** menu for an example). An example of the CCL for **DIVIDER** objects is shown below.

```
DIVIDER: Tools Divider 1
  Position = 250
  Group = Tools
  OS List = winnt, linux-ia64
END
```

The `Position`, `Group` and `OS List` parameters are the same as those used in `APPLICATION` objects. For details, see [APPLICATION \(p. 14\)](#).

1.2.2. Example: Adding the Windows Calculator

The following CCL is the minimum required to add the Windows calculator to the Ansys TurboGrid Launcher:

```
GROUP: Windows Apps
  Menu Name = Windows
END
APPLICATION: Calc
  Group = Windows Apps
  Command = <windir>\system32\calc.exe
  Toolbar Name = Calc
END
```

Although the parameter `Toolbar Name` is not strictly required, you would end up with a blank toolbar button if it were not set.

Chapter 2: CFX Command Language

- [Introduction \(p. 19\)](#)
- [CCL Syntax \(p. 19\)](#)
- [Object Creation and Deletion \(p. 24\)](#)
- [Ansys CFX Expression Language \(p. 25\)](#)

2.1. Introduction

The CFX Command Language (CCL) is the internal communication and command language of Ansys TurboGrid. It is a simple language that can be used to create objects or perform actions in the Post-processor. All CCL statements can be classified into one of three categories.

1. Object and parameter definitions: CCL object and parameter definitions can be used to create or delete objects. For details, see [Object Creation and Deletion \(p. 24\)](#). A list of all objects and parameters that can be used in Ansys TurboGrid is available in the CCL details chapter in the reference guide.
2. Actions: CCL actions are commands that perform a specific task (for example, reading a session file). For details, see [Command Actions \(p. 27\)](#).
3. Power Syntax: Using the Perl programming language, CCL supports programming through Power Syntax with loops, logic and custom macros (subroutines). With Power Syntax, Perl commands can be embedded into CCL to achieve powerful quantitative results. For details, see [Power Syntax \(p. 43\)](#).

State and session files contain object definitions in CCL. In addition, session files can contain CCL action commands. The CCL written to these files can be viewed and modified in a text editor. You can also use a text editor to create your own session and state files to read into Ansys TurboGrid.

Advanced users can interact with Ansys TurboGrid directly through CCL by entering it in the **Command Editor** dialog box or by running Ansys TurboGrid in line interface mode. See [Command Editor Command in the TurboGrid User's Guide](#) and [Line Interface Mode \(p. 47\)](#) for details.

2.2. CCL Syntax

The topic(s) in this section include:

- [Basic Terminology \(p. 20\)](#)
- [Simple Syntax Details \(p. 20\)](#)

2.2.1. Basic Terminology

The following is an example of a CCL object defining an isosurface.

```
USER DEFINED:
  ISOSURFACE: Iso1
    Variable = Minimum Face Angle
    Value = 10 [degree]
    Color = 1,0,0
    Transparency = 0.5
  END
END
```

- ISOSURFACE and USER DEFINED are object types
- Iso1 is an object name
- Variable = Minimum Face Angle is a parameter
- Variable is a parameter name
- Minimum Face Angle is a parameter value
- If the object type ISOSURFACE does not need a name it is called a singleton object. Only one object of a given singleton type can exist.

2.2.1.1. The Data Hierarchy

Data is entered via parameters. These are grouped into objects that are stored in a tree structure. Objects may be at the 'Top Level', or within other objects. Objects inside other objects are said to be 'nested'.

```
OBJECT1: outer object name
OBJECT2: inner object name
  name1 = value
  name2 = value
END
```

Objects and parameters may be placed in any order, provided that the information is set prior to being used further down the file. If data is set in one place and modified in another the latter definition overrides the first.

2.2.2. Simple Syntax Details

The following applies to any line that is not a Power Syntax (or action) line. (That is, any line that does not start with a ! or >.)

2.2.2.1. Case Sensitivity

Everything in the file is sensitive to case.

Case sensitivity is not ideal for users typing in many long parameter names, but it is essential for bringing the Ansys CFX Expression Language (CEL) into CCL. This is because some names used to define CCL objects (such as `Machine Data`) are used to construct corresponding CEL names.

For simplicity and consistency, we recommend the following convention is used in the standard code and its documentation:

- **singletons** and **object types** use upper case only
- **parameter names**, and pre-defined **object names**, are mixed case. We try to follow these conventions:
 - Major words start with an upper case letter, while minor words such as prepositions and conjunctions are left in lower case ("Number of Blade Blocks", for example).
 - Case is preserved for familiar names (for variables "k" or "r", or for the abbreviation "RNG", for example).
- user **object names** conventions are left to you to choose.

2.2.2.2. CCL Names Definition

Names of singletons, types of objects, names of objects, and names of parameters all follow the same rules:

- In simple syntax, a CCL name must be at least one character. This first character must be alphabetic; there may be any number of subsequent characters and these can be alphabetic, numeric, space or tab.
- The effect of spaces in CCL names is:
 - Spaces appearing before or after a name are not considered to be part of the name.
 - Single spaces appearing inside a name are significant.
 - Multiple spaces and tabs appearing inside a name are treated as a single space.

2.2.2.3. Indentation

Nothing in the file is sensitive to indentation. The indentation is used, however, when displaying contents of the file for easier reading.

2.2.2.4. End of Line Comment Character

The # character is used for commenting. Any text to the right of this character is treated as a comment. Any characters may be used within comments.

2.2.2.5. Continuation Character

If a line ends with the character \ the following line is linked to the existing line. There is no restriction on the number of continuation lines.

2.2.2.6. Named Objects

A named object consists of an object type at the start of a line, followed by a `:` followed by an object name. Subsequent lines may define parameters and child objects associated with this object. The object definition is terminated by the string `"END"` on a line by itself.

Object names must be unique within the given scope, and the name must not contain an underscore.

2.2.2.7. Singleton Objects

A singleton object consists of an object type at the start of a line, followed by a `:`. Subsequent lines may define parameters and child objects associated with this object. The object definition is terminated by the string `"END"` on a line by itself.

The difference between a singleton object and a named object is that (after the data has been processed), a singleton can appear just once as the child of a parent object, whereas there may be several instances of a named object of the same type defined with different names.

2.2.2.8. Parameters

A parameter consists of a parameter name at the start of a line, followed by an `=`, followed by a parameter value. A parameter may belong to many different object types. For example `Transparency = 0.6` may belong to a hub geometry object and `Transparency = 0.0` may belong to a volume mesh analysis object. Both refer to the same definition of transparency in the rules file.

2.2.2.9. Lists

Lists are used within the context of parameter values and are comma separated.

2.2.2.10. Parameter values

All parameter values are initially handled as data of type string, and should first of all conform to the following definition of permitted string values:

2.2.2.10.1. String

- Any characters can be used in a parameter value.
- String values or other parameter type values are normally unquoted. If any quotes are present, they are considered part of the value. Leading and trailing spaces are ignored. Internal spaces in parameter values are preserved as given, although a given application is free to subsequently assume a space condensation rule when using the data.
- The characters `$` and `#` have a special meaning. A string beginning with `$` is evaluated as a Power Syntax variable, even if it occurs within a simple syntax statement. This is useful for performing more complex Power Syntax variable manipulation, and then using the result as part of a parameter or object definition. The appearance of `#` anywhere in the CCL file denotes the start of a comment.

- The characters such as [,], { and } are special only if used in conjunction with \$. Following a \$, such characters terminate the preceding Perl variable name.
- Other characters that might be special elsewhere in Power Syntax are escaped automatically when they appear in parameter values. For example, @, % and & are escaped automatically.
- Parameter values can contain commas, but if the string is processed as a list or part of a list then the commas may be interpreted as separators (see below under list data types).

Some examples of valid parameter values using special characters in Power Syntax are:

```
Estimated cost = \$500
Title = Run\#1
Sys Command = "echo 'Starting up Stress solver' ; fred.exe &"
Temporary = $myArray[4]
Option = $myHash{"foo"}
Fuel = C${numberCatoms}H${numberHatoms}
```

Parameter values for data types other than string additionally conform to one of the following definitions.

2.2.2.10.2. String List

A list of string items separated by commas. Items in a string list should NOT contain a comma unless contained between parentheses. One exception can be made if the string list to be is interpreted as a Real List (see below). Otherwise each item in the string list follows the same rules as string data. Example usage:

```
names = one, two, three, four
```

2.2.2.10.3. Integer

Sequence of digits containing no spaces or commas. If a real is specified when an integer is needed the real is rounded to the nearest integer. Example usage:

```
b = 32
```

2.2.2.10.4. Integer List

List of integers, comma separated. Example usage:

```
numbers = 52, 65001, 2
```

2.2.2.10.5. Real

A single precision real number that may be specified in integer, floating point or scientific format, followed optionally by a dimension. Units use the same syntax as CEL.

Expressions can include commas inside function call argument lists. Example usage:

```
a = 12.24
```

```
a = 1.224E01  
a = 12.24 [m s^-1]
```

A real may also be specified as an Expression such as:

```
a = myvel^2 + b  
a = max(b,2.0)
```

2.2.2.10.6. Real List

List of reals, comma separated. Note that all items in the list must have the same dimensions. Those items that are expressions can include commas inside function call argument lists, and the enclosed commas are ignored when the list is parsed into individual items. Example usage:

```
a = 1.0 [m/s], 2.0 [m/s], 3.0 [m/s], 2.0*myvel, 4.0 [cm/s]
```

2.2.2.10.7. Logical

Several forms are acceptable: YES or TRUE or 1 or ON are all equivalent; NO or FALSE or 0 or OFF are all equivalent; initial letter variants Y, T, N, F are accepted (O is not accepted for On/Off); all case variants are accepted. The preferred form, recommended for GUI output files and for user documentation is, Yes/No. Logical strings are also case insensitive (YeS/ nO). Example usage:

```
answer = 1
```

2.2.2.10.8. Logical List

List of Logicals, comma separated. Example usage:

```
answers = oN, YES, 0, fALse, truE
```

2.2.2.11. Escape Character

The \ character is used as an escape character so characters like \$ or # can be used in strings, for example.

2.3. Object Creation and Deletion

You can create objects in Ansys TurboGrid by entering the CCL definition of the object into the **Command Editor** dialog box, or by reading the object definition from a session or state file. The object is created and any associated graphics shown in the viewer. For a list of valid CCL objects, see the CCL details chapter in the reference guide.

You can modify an existing object by entering the object definition with the modified parameter settings into the **Command Editor** dialog box. Only those parameters that are to be changed need to be entered. All other parameters remain unchanged.

There may be a significant degree of interaction between objects in Ansys TurboGrid. For example, a contour plot may depend on the location of an underlying plane, or an isosurface may depend on the

definition of a CEL expression. If changes to one object affect other objects, the other objects are updated automatically.

To delete an object, type `>delete <ObjectPath>`. If you delete an object that is used by other objects, warnings result, but the object is deleted.

An object at the top level can be deleted by specifying its name. To delete a nested object you must enter the path of the object you want to delete. An object's path has the syntax:

```
/parentObjType:parentObjName/.../objType:objName
```

Singletons must be specified in paths as

```
/SINGLETON:SINGLETON
```

For example, to delete an isosurface you have created called Isosurface 1, enter

```
/USER_DEFINED:USER_DEFINED/ISOSURFACE:Isosurface 1
```

2.4. Ansys CFX Expression Language

The Ansys CFX Expression Language (CEL) is integrated into Ansys TurboGrid. You can use an expression defined with CEL in place of any number in TurboGrid. Within TurboGrid you can:

- Create new expressions.
- Set any numeric parameter in an TurboGrid object based on an expression (and the object updates if the expression result changes).
- Create user-defined variables from expressions.
- Directly use the quantitative functions in an expression.
- Specify units as part of an expression.

All expressions are defined in the EXPRESSIONS singleton object. Each expression is a simple `name = expressionstatement` within that object. New expressions are added by defining new parameters within the EXPRESSIONS object (the EXPRESSIONS object is special, in that it does not have a pre-defined list of valid parameters).

Note:

TurboGrid evaluates CEL expressions with single (not double) precision.

Important:

Since Power Syntax uses Perl mathematical operators, you should exercise caution when combining CEL with Power Syntax expressions. For example, in CEL, 2^2 is represented as `2^2`, but in Perl, would be written `2**2`. If you are unsure about the validity of an operator in Perl, consult a Perl reference guide.

2.4.1. CEL Functions, Constants and System Variables

2.4.1.1. CEL Standard Functions

The following is a list of standard functions that are available in Ansys TurboGrid.

Note:

[] denotes a dimensionless quantity. [a] denotes any dimensions of first operand.

Result	Function	Operands
[]	sin	([radian])
[]	cos	([radian])
[]	tan	([radian])
[radian]	asin	([])
[radian]	acos	([])
[radian]	atan	([])
[radian]	atan2	([], [])
[]	exp	([])
[]	loge	([])
[]	log10	([])
[a]	abs	([a])
[a ^{0.5}]	sqrt	([a])
[]	step	([])
[a]	min	([a],[a])
[a]	max	([a],[a])

2.4.1.2. CEL Constants

The following predefined constants can be used within CEL expressions.

Constant	Units	Description
e	<none>	Constant: 2.7182817
g	m s ⁻²	Acceleration due to gravity: 9.806
pi	<none>	Constant: 3.1415927
R	J mol ⁻¹ K ⁻¹	Universal Gas Constant: 8.31447

Chapter 3: Command Actions

- [Introduction \(p. 27\)](#)
- [File Operations from the Command Editor Dialog Box \(p. 28\)](#)
- [Quantitative Calculations in the Command Editor Dialog Box \(p. 37\)](#)
- [Other Commands \(p. 42\)](#)

3.1. Introduction

The **Command Editor** dialog box in Ansys TurboGrid can be used to edit or create graphics objects, perform some typical user actions (reading or creating session and state files, for example) and enter Power Syntax. This section describes the typical user actions you can perform from the **Command Editor** dialog box.

For an introduction to the **Command Editor** dialog box see [Command Editor Command in the TurboGrid User's Guide](#).

For details on editing and creating graphics objects using the CFX Command Language in the **Command Editor** dialog box see [CFX Command Language \(p. 19\)](#).

Power Syntax commands are preceded by the ! symbol. For details on using Power Syntax in the **Command Editor** dialog box see [Power Syntax \(p. 43\)](#).

Action statements cause Ansys TurboGrid to undertake a specific task, usually related to the input and output of data from the system. All actions typed into the **Command Editor** dialog box must be preceded with the > symbol. Actions in session files must also be preceded by the > symbol.

When running Ansys TurboGrid in line interface mode, the TG> command prompt is shown in a Windows Command Prompt or UNIX shell. All the actions described in this section, along with some additional commands, can be typed at the command prompt. You do not have to precede commands with the > symbol when running in line interface mode. For information about using line interface mode, see [Line Interface Mode \(p. 47\)](#).

Many actions require additional information to perform their task (the name of a file to load or the type of hardcopy file to create, for example). By default, these actions get the necessary information from a specific associated CCL singleton object. For convenience, some actions accept a few arguments that can be used to optionally override the commonly changed object settings. If multiple arguments for an action are specified, they must be separated by a comma.

3.1.1. Command Actions Example

The >print command saves the viewer image to a file. All the settings for >print are read from the HARDCOPY: singleton object. However, if you want, you can specify the name of the hardcopy

file as an argument to the >print command. The following CCL example demonstrates both of these alternatives.

```
# Define settings for saving a picture
HARDCOPY:
  Hardcopy Format = jpg
  Hardcopy Filename = default.jpg
  Image Scale = 70
  White Background = Off
END
#Create an output file based on the settings in HARDCOPY
>print
#Create an identical output file with a different filename
>print another_file.jpg
```

3.2. File Operations from the Command Editor Dialog Box

The File Operations available from the **Command Editor** dialog box are outlined in the table below with a summary of the function performed by each option.

Command	Description
savestate	Save the current state to a file. See Save State Files (p. 28) .
readstate	Load a state from an existing state file. See Read State Files (p. 30) .
savetopology	Save the current topology to a file. See Save Topology Files (p. 32) .
savemesh	Save the current mesh to a file. See Save Mesh Files (p. 33) .
saveblade	Save the current blade to a file. See Save Blade Files (p. 33) .
session start	Set the name of a new session file and start recording to it. Stop recording it. See Create Session Files (p. 34) .
session stop	
readsession	Load and execute an existing session file. See Read Session Files (p. 36) .
print	Save the image shown in the viewer window to a file. See Create Hardcopy (p. 37) .
tetin	Save the current geometry to a .tetin file. See Export Geometry (p. 37) .

3.2.1. Save State Files

```
>savestate [mode=<none | overwrite>][filename=<filename>]
```

State files can be used to quickly load a previous state into Ansys TurboGrid. State files can be generated manually using a text editor, or from within Ansys TurboGrid by saving a state file. The >savestate command writes the current Ansys TurboGrid state to a file from the **Command Editor** dialog box.

>savestate supports the following options:

- mode = <none | overwrite>

If mode is `none`, the executor creates a new state file, and if the specified file exists, an error is raised. If mode is `overwrite`, the executor creates a new state file, and if the file exists, it is deleted and replaced with the latest state.

- `filename = <filename>`

Specifies the path and name of the file that the state is written to. If no filename is specified, the `STATE` singleton object is queried for the filename. If the `STATE` singleton does not exist, then an error is raised indicating that a filename must be specified.

3.2.1.1. savestate Command Examples

The following are example `>savestate` commands, and the expected results. If a `STATE` singleton exists, the values of the parameters listed after the `>savestate` command replace the values stored in the `STATE` singleton object. For this command, the `filename` command parameter value replaces the `state filename` parameter value in the `STATE` singleton, and the `mode` command parameter value replaces the `savestate mode` parameter value in the `STATE` singleton.

```
> savestate
```

This command writes the current state to the filename specified in the `STATE` singleton. If the mode in the `STATE` singleton is `none`, and the filename exists, an error is returned. If the mode in the `STATE` singleton is `overwrite`, and the filename exists, the existing file is deleted, and the state is written to the file. If the `STATE` singleton does not exist, an error is raised indicating that a filename must be specified.

```
> savestate mode = none
```

This command writes the current state to the file specified in the `STATE` singleton. If the file already exists, an error is raised. If the `STATE` singleton does not exist, an error is raised indicating that a filename must be specified.

```
> savestate mode = overwrite
```

This command writes the current state to the file specified in the `STATE` singleton. If the file already exists, it is deleted, and the current state is saved in its place. If the `STATE` singleton does not exist, an error is raised indicating that a filename must be specified.

```
> savestate filename = mystate.tst
```

This command writes the current state to the `mystate.tst` file. If the `STATE` singleton exists, and the `savestate mode` is set to `none`, and the file already exists, an error is raised. If the `savestate mode` is set to `overwrite`, and the file already exists, the file is deleted, and the current state is saved in its place. If the `STATE` singleton does not exist, then the system assumes a `savestate mode` of `none`, and behaves as described above.

```
> savestate mode = none, filename = mystate.tst
```

This command writes the current state to the `mystate.tst` file. If the file already exists, an error is raised.

```
> savestate mode = overwrite, filename = mystate.tst
```

This command writes the current state to the `mystate.tst` file. If the file already exists it is deleted, and the current state is saved in its place.

3.2.2. Read State Files

```
>readstate [mode=<overwrite | append>][filename=<filename>, load=<true | false>]
```

The `>readstate` command loads an Ansys TurboGrid State from a specified file.

If a `DATA READER` singleton has been stored in the state file, the `load` action loads the contents of the results file. If a state file contains `BOUNDARY` Objects, and the state file is appended to the current state (with no new `DATA READER` Object), some boundaries defined may not be valid for the loaded results. `BOUNDARY` objects that are not valid for the currently loaded results file are culled.

`>readstate` supports the following options:

- `mode = <overwrite | append>`

If `mode` is set to `overwrite`, the executor deletes all the objects that currently exist in the system, and loads the objects saved in the state file. Overwrite mode is the default mode if none is explicitly specified. If `mode` is set to `append`, the executor adds the objects saved in the state file to the objects that already exist in the system. If the mode is set to `append` and the state file contains objects that already exist in the system, the following logic is used to determine the final result: If the system has an equivalent object, that is, name and type, then the object already in the system is modified with the parameters saved in the state file. If the system has an equivalent object in name only, then the object that already exists in the system is deleted, and replaced with that in the state file.

- `filename = <filename>`

The path to the state file.

- `load = <true | false>`

If `load` is set to `true` and a `DATA READER` object is defined in the state file, then the Results file is loaded when the state file is read. If `load` is set to `false`, the results file is not loaded, and the `DATA READER` object that currently is in the object Database (if any) is not updated.

The following table describes the options, and the corresponding action taken on the objects and the `DATA READER`.

Mode Selection	Load Data Selection	What happens to the Objects?	What happens to the DATA READER?
Overwrite	True	All user objects are deleted. The loading of the new results file changes the default objects (boundaries, wireframe, and so on) including deletion of objects that are no longer relevant to the new results. Default objects that are not explicitly modified by object definitions in the state file have all user modifiable values reset to default values.	It is deleted and replaced.
Overwrite	False	All user objects are deleted. All default objects that exist in the state file update the same objects in the current system state if they exist. Default objects in the state file that do not exist in the current state are not created. All user objects in the state file are created.	If it exists, it remains unchanged regardless of what is in the state file.
Append	True	No objects are initially deleted. The default objects in the state file replace the existing default objects. User objects: <ul style="list-style-type: none"> • are created if they have a unique name. • replace existing objects if they have the same name but different type. • update existing objects if they have the same name and type. 	It is modified with the new value from the state file.
Append	False	No objects are initially deleted. Default objects in the state file only overwrite those in the system if they already exist. User objects have the same behavior as the Append/True option above.	If it exists, it remains unchanged regardless of what is in the state file.

3.2.2.1. readstate Command Examples

The following are example >readstate commands, and the expected results. If a STATE singleton exists, the values of the parameters listed after the >readstate command replace the values stored in the STATE singleton object. For this command, the filename command parameter

value replaces the `state filename` parameter value in the `STATE` singleton, and the `mode` command parameter value replaces the `readstate mode` parameter value in the `STATE` singleton.

```
> readstate
```

This command overwrites or appends to the objects in the system using the objects defined in the file referenced by the `state filename` parameter in the `STATE` singleton. If the `STATE` singleton does not exist, an error is raised indicating that a filename must be specified.

```
> readstate filename = mystate.tst
```

The `readstate mode` parameter in the `STATE` singleton determines if the current objects in the system are deleted before the objects defined in the `mystate.tst` file are loaded into the system. If the `STATE` singleton does not exist, then the system objects are deleted before loading the new state information.

```
> readstate mode = overwrite, filename = mystate.tst
```

This command deletes all objects currently in the system, opens the `mystate.tst` file if it exists, and creates the objects as stored in the state file.

```
> readstate mode = append, filename = mystate.tst
```

This command opens the `mystate.tst` file if it exists, and adds the objects defined in the file to those already in the system following the rules specified in the above table.

```
> readstate mode = overwrite
```

This command overwrites the objects in the system with the objects defined in the file referenced by the `state filename` parameter in the `STATE` singleton. If the `STATE` singleton does not exist, an error is raised indicating that a filename must be specified.

```
> readstate mode = append
```

This command appends to the objects in the system using the objects defined in the file referenced by the `state filename` parameter in the `STATE` singleton. If the `STATE` singleton does not exist, an error is raised indicating that a filename must be specified.

3.2.3. Save Topology Files

```
>savetopology [filename=<filename>]
```

Topology files can be used to easily use a previously defined topology in Ansys TurboGrid. Topology files can be generated manually using a text editor, or from within Ansys TurboGrid by saving a topology file. The `>savetopology` command writes the current Ansys TurboGrid Topology to a file from the **Command Editor** dialog box.

>savetopology requires the following argument:

- filename = <filename> Specifies the path and name of the file that the topology is written to.

3.2.3.1. savetopology Command Example

The following is an example >savetopology command, and the expected results.

```
> savetopology filename = mytopology.tgt
```

This command writes the current topology to the mytopology.tgt file.

3.2.4. Save Mesh Files

```
>savemesh [filename=<filename>, solver=<cfx5 | tascflow>]]
```

Mesh files can be used to load a previously created mesh into Ansys TurboGrid and export a completed mesh to be used in an Ansys CFX Solver. The >savemesh command writes the current Ansys TurboGrid Mesh to a file from the **Command Editor** dialog box.

>savemesh requires the following arguments:

- filename = <filename>

Specifies the path and name of the file to which the state is written.

- solver = <cfx5 | tascflow>

If solver is cfx5, the mesh file is saved in the Ansys CFX mesh format (.gtm). If solver is tascflow, the mesh file is saved in CFX-TASCflow format, including a .grd and .bcf file.

3.2.4.1. savemesh Command Examples

The following is an example >savemesh command, and the expected results.

```
> savemesh filename = mymesh.gtm, solver = cfx5
```

The command above writes the current mesh to mymesh.gtm in Ansys CFX format.

```
> savemesh filename = mymesh, solver = tascflow
```

The command above writes the current Mesh to mymesh.grd and mymesh.bcf in CFX-TASCflow format.

3.2.5. Save Blade Files

```
>saveblade [filename=<filename>]
```

Blade files can be used to save the changes made to the geometry. The `>saveblade` command writes the current Ansys TurboGrid Blade to a file from the **Command Editor** dialog box.

`>saveblade` requires the following argument:

- `filename = <filename>`

Specifies the path and name of the file to which the blade is written.

3.2.5.1. saveblade Command Example

The following is an example `>saveblade` command, and the expected results.

```
> savetopology filename = mytopology.tgt
```

This command writes the current topology to the `mytopology.tgt` file.

3.2.6. Create Session Files

First, it is necessary to set the name of the file to which your session commands are to be saved. This can be done by typing the CCL for the singleton object `SESSION`.

In the command window, type:

```
SESSION:
  Session Filename = <filename>.tse
END
```

To begin recording commands, type the following line into the **Command Editor** dialog box:

```
>session start [mode=<none | overwrite | append>, filename=<filename>]
```

To stop recording commands, type the following line into the **Command Editor** dialog box:

```
>session stop
```

Session files can be used to quickly reproduce all the actions performed in a previous Ansys TurboGrid Session. Session files can be generated manually using a text editor, or from within Ansys TurboGrid by recording a session. The commands required to write to these files from the **Command Editor** dialog box are described below. The `>session` command handles all **Write Session** features.

The following options are available to support the functionality:

`>session start` supports the following options:

- `mode = <none | overwrite | append>`

If `mode` is set to `none` (the default value), an error is raised if the file already exists. If `mode` is set to `overwrite`, the file is deleted and newly created if it already exists. If `mode` is set to `append`, the new session is appended to the end of the existing file.

- `filename = <filename>`

Specifies the filename and path to the session file. If no filename is specified, the SESSION singleton indicates the filename and the mode to use for overwriting/appending. If no SESSION singleton exists, an error is raised indicating that a filename must be specified.

The `session stop` command terminates the saving of a session, and closes the session file. No options are accepted with this option.

3.2.6.1. session Command Examples

The following are example `>session` commands, and the expected results. If a SESSION singleton exists, the values of the parameters listed after the `>session` command replace the values stored in the SESSION singleton object. For this command, the `filename` parameter value replaces the `session filename` parameter value in the SESSION singleton, and the `mode` command parameter value replaces the `write session mode` parameter value in the SESSION singleton.

```
> session start, filename = mysession.tse
```

This action starts a new session in a filename called `mysession.tse`. If `mysession.tse` already exists, the overwrite/append behavior is dependent on that set in the SESSION singleton. If no SESSION singleton exists, and the `mysession.tse` file also exists, the command fails with an error message (that is, default mode is none).

```
> session start, mode = none, filename = mysession.tse
```

This command starts a new session file in a filename called `mysession.tse`. If `mysession.tse` already exists, the command fails with an error message.

```
> session start, mode = overwrite, filename = mysession.tse
```

This command starts a new session. If `mysession.tse` already exists it is deleted, and replaced with the new session. If the file does not already exist, it is created.

```
> session start, mode = append, filename = mysession.tse
```

This command starts a new session. If `mysession.tse` already exists, the new session is appended to the end of the existing file. If the file does not already exist, it is created.

```
> session start
```

This command starts a new session, using the mode and filename defined in the SESSION singleton. If the SESSION singleton does not exist, an error message is raised indicating that a filename must be specified.

```
> session start, mode = overwrite
```

This command starts a new session with a filename of that specified in the `SESSION` singleton. If the specified file already exists, it is deleted and a new file is created. If the `SESSION` singleton does not exist, an error message is raised indicating that a filename must be specified.

```
> session start, mode = append
```

This command starts a new session with a filename of that specified in the `SESSION` singleton. If the specified file already exists, the new session is appended to the end of the existing file. If the `SESSION` singleton does not exist, an error message is raised indicating that a filename must be specified.

```
> session start, mode = none
```

This command starts a new session with a filename of that specified in the `SESSION` singleton. If the specified file already exists, the command is terminated with an error message. If the `SESSION` singleton does not exist, an error message is raised indicating that a filename must be specified.

```
> session stop
```

This command terminates the current session in progress, and closes the currently open session file.

3.2.7. Read Session Files

```
>readsession [filename=<filename>]
```

The `>readsession` command performs session file reading and executing.

`<readsession` supports the following options:

- `filename = <filename>`

Specifies the filename and path to the session file that should be read and executed. If no filename is specified, the `SESSION` singleton object indicates the file to use. If no `SESSION` singleton exists, an error is raised indicating that a filename must be specified.

3.2.7.1. readsession Command Examples

The following are example `>readsession` commands, and the expected results. If a `SESSION` singleton exists, the values of the parameters listed after the `>readsession` command replace the values stored in the `SESSION` singleton object. For this command, the `filename` command parameter value replaces the `session filename` parameter value in the `SESSION` singleton.

```
> readsession
```

This command reads the session file specified in the `SESSION` singleton, and executes its contents. If the `SESSION` object does not exist, an error is raised indicating that a filename must be specified.

```
> readsession filename = mysession.tse
```

This command reads and executes the contents of the `mysession.tse` file.

3.2.8. Create Hardcopy

```
>print [<filename>]
```

The `>print` command creates an image file showing the viewer contents. Settings for output format, quality, and so on, are read from the `HARDCOPY` singleton object. For an example of the `>print` command, see [Command Actions Example \(p. 27\)](#).

The optional argument `<filename>` can be used to specify the name of the output file to override that stored in `HARDCOPY`.

Note:

The `HARDCOPY` singleton object must exist before the `>print` command is executed.

3.2.9. Export Geometry

The `>tetin` command exports the current geometry to a Tetin (`.tin`) file that can be read into ICEM CFD products.

`>tetin` requires the following argument:

- `filename = <filename>`

Specifies the filename and path to which the Tetin file is written.

3.2.9.1. tetin Command Example

The following is an example `>tetin` command, and the expected results.

```
> tetin filename = mytetin.tin
```

This command writes the current geometry to the `mytetin.tin` file.

3.3. Quantitative Calculations in the Command Editor Dialog Box

When executing a calculation from the **Command Editor** dialog box, the result is displayed in the built-in calculator.

The `>calculate` command is used to perform function calculations in the **Command Editor** dialog box. Typing `>calculate` alone performs the calculation using the parameters stored in the `CALCULATOR` singleton object. Typing `>calculate <function name>` does not work if required arguments are needed by the function.

3.3.1. Function Calculation

```
>calculate [<function name>, <arguments>]
```

A number of useful quantitative functions have been defined within Ansys TurboGrid. These functions are specified in the table below. You can follow the links in the **Function Name** column to see the syntax used for each function when using them from the **Command Editor** dialog box. A detailed description of the calculation performed by each function is not provided here. For details, see [Function in the TurboGrid User's Guide](#). When using quantitative functions as part of a CEL expression, a different syntax is required. For details, see [CEL Standard Functions \(p. 26\)](#).

These functions may be accessed from the **Command Editor** dialog box using the `>calculate` command, or via function calls within Power Syntax. Reference documentation on the Power Syntax functions is provided in [Power Syntax \(p. 43\)](#).

When a function is evaluated, a singleton CALCULATOR object is created that shows the parameters and results for the most recent calculation. If the `>calculate` command is supplied without `<function name>` or `<arguments>`, then the calculation is performed with the settings from the CALCULATOR object (which may be modified like any other object in the **Command Editor** dialog box). An error is raised if the CALCULATOR object does not exist.

When evaluated as part of a Power Syntax function, the results of the calculation can be stored in a Power Syntax variable for further processing. See [Examples of Power Syntax \(p. 44\)](#) for examples of using calculations within Power Syntax.

Function Name	Operation	Arguments <required> [<optional>]
area (p. 39)	Area projected to axis (no axis specification returns total area)	<Location>, [<Axis>]
areaAve (p. 40)	Area-weighted average	<Expression>, <Location>, [<Axis>]
areaInt (p. 40)	Area-weighted integral	<Expression>, <Location>, [<Axis>]
ave (p. 40)	Arithmetic average	<Expression>, <Location>
count (p. 40)	Number of calculation points	<Location>
length (p. 40)	Length of a curve	<Location>
lengthAve (p. 40)	Length-weighted average	<Expression>, <Location>
lengthInt (p. 41)	Length-weighted integration	<Expression>, <Location>
maxVal (p. 41)	Maximum Value	<Expression>, <Location>

Function Name	Operation	Arguments <required> [<optional>]
minVal (p. 41)	Minimum Value	<Expression>, <Location>
probe (p. 41)	Value at a point	<Expression>, <Location>
sum (p. 41)	Sum over the calculation points	<Expression>, <Location>
volume (p. 41)	Volume of a 3D location	<Location>
volumeAve (p. 41)	Volume-weighted average	<Expression>, <Location>
volumeInt (p. 41)	Volume-weighted integral	<Expression>, <Location>

Note:

The following functions are not available in Ansys TurboGrid: force, forceNorm, massFlow, massFlowAve, massFlowInt and torque.

3.3.1.1. Expression Specification

Any variable or valid expression can be used as the <Expression> argument. For example, `areaAve Minimum Face Angle, myPlane` is allowed. For a list of variables and valid expressions, see [CEL Functions, Constants and System Variables \(p. 26\)](#).

3.3.1.2. Axis Specification

Some functions take an axis specification as an argument. The general format for an axis specification is:

```
<x|y|z>
```

3.3.1.3. Quantitative Function List**3.3.1.3.1. area**

```
>calculate area, <Location>, [<Axis>]
```

The specification of an axis is optional. If one is not specified, the value held in the CALCULATOR object is used. To calculate the total area of the location, the axis specification should be left blank (you should type a comma after the location specification). See [area in the TurboGrid User's Guide](#) for a detailed function description.

Example: >calculate area, myplane calculates the area of the locator myplane projected onto a plane normal to the axis specification in the CALCULATOR object.

Example: >calculate area, myplane, calculates the area of the locator myplane. Note that adding the comma after myplane removes the axis specification.

3.3.1.3.2. areaAve

```
>calculate areaAve, <Expression>, <Location>, <Axis>
```

See [areaAve in the TurboGrid User's Guide](#) for a detailed function description.

3.3.1.3.3. areaInt

```
>calculate areaInt, <Expression>, <Location>, [<Axis>]
```

Axis is optional. If it is not specified the value held in the CALCULATOR object is used. To perform the integration over the total face area, the axis specification should be blank (you should type a comma after the location name). See [areaInt in the TurboGrid User's Guide](#) for a detailed function description.

3.3.1.3.4. ave

```
>calculate ave, <Expression>, <Location>
```

See [ave in the TurboGrid User's Guide](#) for a detailed function description.

3.3.1.3.5. count

```
>calculate count, <Location>
```

See [count in the TurboGrid User's Guide](#) for a detailed function description.

3.3.1.3.6. length

```
>calculate length, <Location>
```

Note:

When using this function in Power Syntax the leading character should be capitalized to avoid confusion with the Perl internal command "length". See [length in the TurboGrid User's Guide](#) for a function description.

3.3.1.3.7. lengthAve

```
>calculate lengthAve, <Expression>, <Location>
```

See [lengthAve](#) in the *TurboGrid User's Guide* for a detailed function description.

3.3.1.3.8. lengthInt

```
>calculate lengthInt, <Expression>, <Location>.
```

See [lengthInt](#) in the *TurboGrid User's Guide* for a detailed function description.

3.3.1.3.9. maxVal

```
>calculate maxVal, <Expression>, <Location>
```

See [maxVal](#) in the *TurboGrid User's Guide* for a detailed function description.

3.3.1.3.10. minVal

```
>calculate minVal, <Expression>, <Location>
```

See [minVal](#) in the *TurboGrid User's Guide* for a detailed function description.

3.3.1.3.11. probe

```
>calculate probe, <Expression>, <Location>
```

See [probe](#) in the *TurboGrid User's Guide* for a detailed function description.

3.3.1.3.12. sum

```
>calculate sum, <Expression>, <Location>
```

See [sum](#) in the *TurboGrid User's Guide* for a detailed function description.

3.3.1.3.13. volume

```
>calculate volume, <Location>
```

See [volume](#) in the *TurboGrid User's Guide* for a detailed function description.

3.3.1.3.14. volumeAve

```
>calculate volumeAve, <Expression>, <Location>
```

See [volumeAve](#) in the *TurboGrid User's Guide* for a detailed function description.

3.3.1.3.15. volumeInt

```
>calculate volumeInt, <Expression>, <Location>
```

See [volumeInt](#) in the *TurboGrid User's Guide* for a detailed function description.

3.4. Other Commands

3.4.1. Deleting Objects

```
>delete <objectnamelist>
```

The `>delete` command can be used in the **Command Editor** dialog box to delete objects. The command must be supplied with a list of object names separated by commas. An error message is displayed if the list contains any invalid object names, but the deletion of valid objects in the list is still processed.

3.4.2. Viewing a Chart

```
>chart <objectname>
```

The `>chart` command is used to invoke the **Chart Viewer** and display the specified CHART object. Chart objects and Chart Lines are created like other CCL objects.

3.4.3. Creating a Mesh

```
>mesh
```

The `>mesh` command is used in the **Command Editor** dialog box to create a mesh using the current Topology and Mesh Data objects.

Chapter 4: Power Syntax

- [Introduction \(p. 43\)](#)
- [Examples of Power Syntax \(p. 44\)](#)
- [Predefined Power Syntax Subroutines \(p. 45\)](#)

4.1. Introduction

Programming constructs can be used within CCL for advanced usage. Rather than invent a new language, CCL takes advantage of the full range of capabilities and resources from an existing programming language, Perl. Perl statements can be embedded between lines of simple syntax, providing capabilities such as loops, logic, and much, much more with any CCL input file.

A line of Power Syntax is identified in a CCL file by an exclamation mark (!) in the first column of a line. In between Perl lines, simple syntax lines may refer to Perl variables and lists. Examples of CCL with Power Syntax can be found in [Examples of Power Syntax \(p. 44\)](#).

A wide range of additional functionality is made available to expert users with the use of Power Syntax including:

- Loops
- Logic and control structures
- Lists and arrays
- Subroutines with argument handling (useful for defining commonly re-used plots and procedures)
- Basic I/O and input processing
- System functions
- much, much more (Object programming, WorldWide Web access, simple embedded GUIs).

Any of the above may be included in a CCL input file or Ansys TurboGrid Session file.

Important:

You should be wary when entering certain expressions, since Power Syntax uses Perl mathematical operators. For example, in CEL, 2^2 is represented as 2^2 , but in Perl, would be written $2**2$. If you are unsure about the validity of an operator, you should check a Perl reference guide.

There are many good reference books on Perl. Two examples are "Learning Perl" (ISBN 1-56592-042-2) and "Programming Perl" (ISBN 1-56592-149-6) from the O'Reilly series.

4.2. Examples of Power Syntax

The following are some examples in which the versatility of Power Syntax is demonstrated. They become steadily more complex in the latter examples.

All arguments passed to subroutines should be enclosed in quotations, for example `Plane 1` must be passed as `"Plane 1"`, and `Minimum Face Angle` should be entered as `"Minimum Face Angle"`. Any legal CFX Command Language characters that are illegal in Perl need to be enclosed in quotation marks.

4.2.1. Example 1: Using a for Loop

This example demonstrates using Power Syntax to wrap a `for` loop around some CCL Object definitions to repetitively change the visibility on the outer boundaries.

```
# Make the hub and shroud surfaces gradually transparent in
# the specified number of steps.
!$numsteps = 10;
!for ($i=0; $i < $numsteps; $i++) {
  ! $trans = ($i+1)/$numsteps;
  GEOMETRY:
    HUB:
      Visibility = 1
      Transparency = $trans
    END
  END
  GEOMETRY:
    SHROUD:
      Visibility = 1
      Transparency = $trans
    END
  END
!}
```

The first line of Power Syntax defines a scalar variable called `numsteps`. Scalar variables (that is, simple single-valued variables) begin with a "\$" symbol in Perl. The next line defines a `for` loop that increments the variable `i` up to `numsteps`. Next, we determine the fraction we are along in the loop and assign it to the variable `trans`. The object definitions then use `trans` to set their transparency and then repeat. Note how Perl variables can be directly embedded into the object definitions. The final line of Power Syntax (`! }`) closes the `for` loop.

4.2.2. Example 2: Creating a Simple Subroutine

This example defines a simple subroutine to make two `TURBO SURFACE` objects at specified locations. The subroutine is used in the next example.

```
!sub makeTurboSurfaces {
  USER DEFINED:
    TURBO SURFACE:Span Surface
      Draw Faces = Off
      Draw Lines = On
      Variable = Span Normalized
      Value = 0.25
```

```

    Visibility = On
END

TURBO SURFACE:Theta Surface
    Draw Faces = Off
    Draw Lines = On
    Variable = Theta
    Value = 300 [degree]
    Visibility = On
END
END
!}

```

You can execute this subroutine by typing `!makeTurboSurfaces()` in the **Command Editor** dialog box.

4.3. Predefined Power Syntax Subroutines

This section contains subroutines to provide additional Power Syntax functionality in Ansys TurboGrid. You can view a list of these subroutines by entering `!showSubs()` in the **Command Editor** dialog box. The list is printed to the console window. The list shows all currently loaded subroutines, so it includes any custom subroutines that you have processed in the **Command Editor** dialog box.

4.3.1. evaluate(Expression)

```
real,string evaluate("Expression")
```

A utility function that takes an expression and returns the evaluated expression. The returned value is a list variable in which the first element is the numeric value of the expression, and the second is the base unit. For example:

```

!@myexp = evaluate("10 [degree]/3.0");
!print "myexp=", $myexp[0], " [", $myexp[1], "]\n";

```

prints:

```
myexp=0.0581776648759842[rad]
```

4.3.2. getValue(Object Name, Parameter Name)

A utility function that takes a name or path of a CCL object and parameter name and returns the value of the parameter. For example,

```
! $bladecount = getValue("/GEOMETRY:GEOMETRY/MACHINE DATA:MACHINE DATA", "Bladeset Count");
```

4.3.3. showPkgs()

```
void showPkgs()
```

A content subroutine that prints a list of available packages that may contain other variables or subroutines in Power Syntax.

4.3.4. showSubs()

```
void showSubs("String packageName")
```

A content subroutine that prints a list of the subroutines available in the specified package. If no package is specified, Ansys TurboGrid is used by default, which lists the routines specified here.

4.3.5. showVars()

```
void showVars("String packageName")
```

A content subroutine that prints a list of the Power Syntax variables and their current value defined in the specified package. If no package is specified, Ansys TurboGrid is used by default.

4.3.6. verboseOn()

Returns 1 or 0 depending if the Perl variable `$verbose` is set to 1.

Chapter 5: Line Interface Mode

- [Introduction \(p. 47\)](#)
- [Line Interface Mode \(p. 48\)](#)
- [Batch Mode \(p. 50\)](#)

5.1. Introduction

Some user interaction (for example with some advanced features) with Ansys TurboGrid is through the CFX command line. This is denoted by the "TG>" command line prompt. At the command line prompt you can issue CFX Command Language (CCL) actions, create CCL objects, and issue a few command-line-specific commands. For further information on creating CCL objects, see the CCL details chapter in the reference guide.

By default, any entry on the command line is assumed to be a CCL action, and is immediately processed by Ansys TurboGrid. The table below provides a complete list of special commands that are accepted by Ansys TurboGrid, but are not actually CCL actions. Refer to [Command Actions \(p. 27\)](#) for information on CCL actions that can be used on the command line.

Action (Abbreviation)	Arguments	Behavior
help (h)	<i>none</i>	Lists all valid command-line and CCL actions.
getstate (s)	<Object Name>	Without an argument, lists all currently defined objects. With an argument, shows details of the definition of the named object.
enterccl (e)	<i>none</i>	Enters CCL object-definition mode. Enables you to type CCL object definitions. Ctrl+e processes the object definition, Ctrl+x aborts.
!	<command>	Executes a Power Syntax (Perl) command
%	<command>	Executes a system command (Unix Only)

Because a full mesh creation would require a large amount of typing, and because of the flexibility of the CCL Power Syntax, most input will be via session files. A session file can be specified at start-up via

the `-session <filename>` option, or on the Ansys TurboGrid command line using the `readsession` command. For further information on the `readsession` command, see [Read Session Files \(p. 36\)](#).

5.2. Line Interface Mode

All of the functionality of Ansys TurboGrid can be accessed when running in Line Interface mode. This section contains information on how to perform typical user actions (loading state files, saving pictures, and so on), defining geometry, creating graphical objects and performing quantitative calculations when running Ansys TurboGrid in Line Interface mode.

In Line Interface mode you are typing the CCL commands that would otherwise be issued by the user interface. A Viewer is provided in a separate window which shows the geometry and the objects that you create on the command line.

To run in Line Interface mode (`<CFXROOT>` should be replaced with the filepath to your installation of Ansys TurboGrid):

Windows: Execute the command `<CFXROOT>\bin\cfxtg -line` at the Command Prompt (omitting the `-line` option starts the user interface mode).

You may want to change the size of the Command Prompt window to view the output from commands such as `getstate`. This can be done by typing `mode con lines=X` at the Command Prompt *before* starting Ansys TurboGrid, where X is the number of lines to display in the window. You can choose a large number of lines if you want to be able to see all the output from a session (a scroll bar is used in the Command Prompt window).

Note:

Note that once inside Ansys TurboGrid, file paths should contain *forward* slashes.

UNIX: Execute the command `<CFXROOT>/bin/cfxtg -line` at the command prompt (omitting the `-line` option starts the user interface mode).

In Ansys TurboGrid Line Interface mode, all commands are assumed to be actions, the `>` symbol required in the **Command Editor** dialog box is not needed.

All of the functionality available from the **Command Editor** dialog box in the user interface is available in Line Interface mode by typing `enterccl` or `e` at the command prompt. When in `e` mode you can type any set of valid CCL commands. The commands are not processed until you leave `e` mode by typing `.e`. You can cancel `e` mode without processing the commands by typing `.c`. For details, see [Command Editor Command in the TurboGrid User's Guide](#).

A list of Command Actions and their explanation are described in the Command Actions documentation. For details, see [Introduction \(p. 27\)](#).

Note:

The action commands shown in the Command Actions documentation are preceded by the `>` symbol. This symbol should be omitted when typing action commands at the command prompt.)

You can create objects by typing the CCL definition of the object when in `e` mode, or by reading the object definition from a session or state file. See [File Operations from the Command Editor Dialog Box \(p. 28\)](#) for details.

Line Interface mode differs from the **Command Editor** dialog box in that Line Interface action commands are not preceded by the ">" symbol.

All commands that work for the **Command Editor** dialog box also work in line interface mode, providing the correct syntax is used.

Some commands specific to line interface mode are listed below.

5.2.1. Lists of Commands

To show a list of valid commands, type `help` at the command prompt.

To see help for all CCL actions, refer to [Command Actions \(p. 27\)](#).

5.2.2. Viewer Hotkeys

The zoom, rotate, pan, and other mouse actions available for manipulating the viewer in user interface mode perform identical functions in the viewer in line interface mode. In addition to this, hotkeys can be used to manipulate other aspects of the viewer. For a full list of all the hotkeys available click in the viewer to make it the active window and select the ? icon. To execute a hotkey command, click once in the viewer (or on the object, as some functions are object-specific) and type the command.

5.2.3. Calculator

When evaluated on the command line, the result of a calculation is printed to standard output.

For a list of valid calculator functions and required parameters, type `calculate help` at the command prompt. For details, see [Quantitative Calculations in the Command Editor Dialog Box \(p. 37\)](#).

5.2.4. getstate Command

The list of all currently defined objects can be obtained using the `getstate` command. To get details on a specific object, type `getstate <ObjectName>`.

5.2.5. Repeating CCL Commands

If you want to carry out the last CCL command again, type `=`.

5.2.6. Executing a Shell Command

If you want to carry out a shell command, type `%` directly before your command. For example, `%ls` lists all the files in your current directory.

5.2.7. Quitting

Ansys TurboGrid can be shut down by typing the `quit` command at the command prompt.

5.2.8. Example

The following example provides a set of commands that you could type at the TG> command prompt. The output that is written to the screen when executing these commands is not shown.

```
TG> getstate /GEOMETRY:GEOMETRY/HUB:HUB
TG> e
GEOMETRY:
HUB:
  Visibility = On
  Transparency = 0.1
END
END
.e
TG> getstate /GEOMETRY:GEOMETRY/HUB:HUB
TG> quit
```

5.3. Batch Mode

All of the functionality of Ansys TurboGrid can be accessed when running in batch mode. When running in batch mode, a viewer window is not provided and you cannot type commands at a command prompt.

Commands are issued via an Ansys TurboGrid Session file (*.tse), the name of which is specified when executing the command to start batch mode.

To run in batch mode, execute the following command at the command prompt:

<i>Windows</i>	<code><CFXROOT>\bin\cfxtg -batch <file-name>.tse</code>
<i>UNIX</i>	<code><CFXROOT>/bin/cfxtg -batch <file-name>.tse</code>

The session file can be created using a text editor, or, more easily, by recording a session file while running in Line Interface or user interface mode.

The last command of any session file used in batch mode should be >quit. If this is not the case, you will become locked in batch mode and have to terminate Ansys TurboGrid manually. If a session file is recording when you quit from user interface or line interface mode, the >quit command will be automatically written to your session file before Ansys TurboGrid quits. Alternatively, you can use a text editor to add this command to the end of a session file if you want Ansys TurboGrid to quit after playing the session file.

5.3.1. Example: Generating a Similar Mesh from Different Curve Files

As mentioned above, the most common approach to generating session files for use in batch processing is to record sessions carried out interactively. For example, you may record a session file of an entire session in which you load profile points, generate topology, create a mesh and save it to a .gtm file. If you then edit the session file (using a text editor) replacing the hub, shroud, blade and mesh files with other filenames, you can then repeat the same actions on different curve files by running the edited session file.

You may also edit a session file to include a loop written with Power Syntax (Perl script). The session file could load a state file before entering the loop so that only the CCL block structure that controls

the parameters that vary must be in the loop. To obtain a particular CCL block structure, you can do one of the following:

1. Write a state file and then use a text editor to select the appropriate CCL block.
2. Write a state file, saving only the CCL blocks that you need.
3. Begin recording a session file. Click **Apply** on the object editor for the object of interest. Stop recording the session file. Open the session file in a text editor and extract the appropriate CCL block.

The following is a simplified example of a loop that could be used in a parametric study. It loads 4 different blade files, blade1.curve, blade2.curve... and so on, and writes 4 corresponding mesh files, mesh1.gtm, mesh2.gtm... while keeping all other settings the same.

```
! for (1..4)
! {
! my $bladeFile = "blade"._.".curve";
! my $meshFile = "mesh"._.".gtm";
GEOMETRY:GEOMETRY
  BLADE:Blade 1
    Coordinate Frame Type = Cartesian
    Curve Representation = Bspline
    Input Angle Units = degree
    Input Filename = $bladeFile
    Input Length Units = m
    Show Curve = Off
    Show Surface = On
    Surface Representation = Bspline
    Visibility = On
  END
END
> mesh
> savemesh filename=$meshFile, onefile=On, solver=cfx5
! }
```

The next example illustrates how a numerical parameter can be modified in a batch loop. It makes use of a list of values.

```
! for (20,27.5,35,42.5)
! {
!
! my $hubAngle = $_. " [degree]";
GEOMETRY:GEOMETRY
  INLET:
    Hub Angle = $hubAngle
    Override Automatic Angles = On
    Shroud Angle = 0.0 [degree]
    Visibility = Off
  GEO POINT:Low Hub Point
    Requested ART = -58.09,166.585,54.1607
    Visibility = On
  END
  GEO POINT:Low Shroud Point
    Visibility = On
  END
END
END
! }
```

A hash table may be used to map integers to numbers or text strings. The following example makes use of a hash table to map from the loop index (an integer) to various real numbers:

```
! my %hubAngle=
! (
!   1 => 20,
!   2 => 27.5,
!   3 => 35,
!   4 => 42.5,
! );
! for (1..4)
! {
!
! my $hubAngleVal = $hubAngle{$_}." [degree]";
GEOMETRY:GEOMETRY
  INLET:
    Hub Angle = $hubAngleVal
    Override Automatic Angles = On
    Shroud Angle = 0.0 [degree]
    Visibility = Off
  GEO POINT:Low Hub Point
    Requested ART = -58.09,166.585,54.1607
    Visibility = On
  END
  GEO POINT:Low Shroud Point
    Visibility = On
  END
END
END
! }
```

A hash table can also control parameters that require string values. An example of a string hash table follows:

```
! my %values =
! (
!   1 => "On",
!   2 => "Off",
!   3 => "lastval"
! );
! my $firstval = $values{1};
```

In this example, `firstval` will be set to `On`. To use this hash table in a loop, the quantity in `{ }` after `$values` could be `$_` which represents the loop index.

Chapter 6: Meshing Reference

Ansys TurboGrid employs the traditional concept of block-structure (multi-block) mesh generation. The block-structure approach is simple and efficient, enabling the use of Transfinite Interpolation (TFI) for surface and interior mesh generation.

Although it is possible to establish a correspondence between any physical region and any given logically rectangular block, the grid inside such a block is likely to be unusable as the geometry becomes more complex. Topology blocks therefore represent contiguous sub-regions of a physical domain. Within each block, the mesh elements are logically rectangular but the blocks themselves fit together in an unstructured manner. Each block has its own curvilinear coordinate system and is logically rectangular. This allows the grid generation and numerical solution on the grid to be constructed to operate in a logically rectangular computational region.

There are many subsets of TFI, such as Lagrangian and Hermitic: a useful reference for such subjects is "The Handbook of Grid Generation" (J.F. Thompson et al., 1999, CRC Press). Ansys TurboGrid employs an algebraic, semi-isogeometric procedure for surface-mesh generation.

