



©2022 ANSYS, Inc.

All Rights Reserved.

Unauthorized use, distribution
or duplication is prohibited.

Scripting in Mechanical 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 2022 R2
July 2022

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

Copyright and Trademark Information

© 2022 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. FLEXIm 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

Scripting Quick Start	1
 Scripting Introduction	3
Mechanical Scripting View	3
Working with the Editor	4
Working with the Shell	9
Autocompletion	10
Snippets	13
Supplied Snippets	13
Snippet Usage Example	15
Snippet Inserter	16
Snippet Creation and Management	17
Keyboard Shortcuts	20
Editor and Shell Keyboard Shortcuts	20
Text Editor Keyboard Shortcuts	21
Line Operation Shortcuts	21
Selection Shortcuts	22
Multi-Cursor Shortcuts	22
Go-To Shortcuts	23
Folding Shortcuts	23
Other Shortcuts	23
Recording APIs	23
Debugging Scripts and Extensions	24
Scope Selection for ACT Extensions	29
Key Usage Concepts	31
Threading	33
Additional Resources	35
Mechanical APIs	37
 Mechanical API Introduction	39
 Mechanical API Notes	41
Mechanical API Migration Notes	41
Mechanical API Known Issues and Limitations	42
 Object Access	45
Property Types	46
Tree	46
Model Objects	49
Accessing and Manipulating the Geometry Object	49
Accessing and Manipulating the Mesh Object	50
Accessing and Manipulating the Connections Object	51
Accessing and Manipulating the Analysis Object	52
Analysis Settings Object	53
Object Traversal	54
Traversing the Geometry	55
Traversing the Mesh	56
Traversing Results	58
Property APIs for Native Tree Objects	60
Details View Parameters	61
Solver Data	61
Getting Global Items for Solver Data	62
Getting Object-Specific Data	62

Getting Object-Specific Data for Imported Objects	64
Boundary Conditions	69
Input and Output Variables	69
Variable Definition Types	69
Setting Input Loading Data Definitions	69
Setting Variable Definition Types	70
Creating a Displacement and Verifying Its Variable Definition Types	71
Changing the Y Component from Free to Discrete	72
Changing the Y Component Back to Free	73
Setting Discrete Values for Variables	73
Controlling the Input Variable	74
Controlling the Output Variable	75
Adding a Load	76
Extracting Min-Max Tabular Data for a Boundary Condition	78
Setting the Direction of a Boundary Condition	80
Worksheets	81
Named Selection Worksheet	81
Defining the Named Selection Worksheet	81
Adding New Criteria to the Named Selection Worksheet	82
Mesh Order Worksheet	84
Defining the Mesh Worksheet	85
Adding New Rows in the Mesh Worksheet	85
Meshing the Named Selections	86
Working with Weld Worksheet	87
Layered Section Worksheet	89
Bushing Joint Worksheet	89
Getting the Bushing Joint Worksheet and Its Properties	90
Getting the Value for a Given Component	90
Condensed Part Worksheet	90
Graphics	93
Setting Graphics View Options	93
Setting the Display Options for Annotations	96
Result Vector Display Options	99
Manipulating Graphics	100
Exporting Graphics	106
Exporting Result or Probe Animations	107
Creating Section Planes	108
Inserting Image Plane	112
Setting Model Lighting Properties	113
Manage Views with Model View Manager	114
Results	117
Adding Results to a Solution Object	117
Accessing Contour Results for an Evaluated Result	118
Accessing Contour Results from an Evaluated Result	119
Accessing Contour Results Scoped to Faces, Elements, or Nodes	129
Accessing Contour Results Scoped to Paths	131
Accessing Contour Results for Surfaces	133
Accessing Contour Results for Beams	135
Limitations of Tabular Data Interface	137
Set Result Display Options	138
Working with Legends	142

Global Legend Settings	142
Result-Specific Legend Settings	143
Standalone Legend Settings	145
Other APIs	149
Mechanical Interface and Ribbon Tab Manipulation	149
Command Snippets	149
Object Tags	150
Solve Process Settings	150
Message Window	151
Ray Casting on Geometry	151
Interacting with Legacy JScript	154
Data Processing Framework	157
Scripting Examples	161
Script Examples for Selection	163
Select Geometry or Mesh in the Graphics Window	163
Get Tree Object of a Body Corresponding to a Selected Body	164
Get GeoData Body Corresponding to a Tree Object of a Body	164
Query Mesh Information for Active Selection	164
Use an Existing Graphics Selection on a Result Object	164
Calculate Sum of Volume, Area, and Length of Scoped Entities	165
Create a Named Selection from the Scoping of a Group of Objects	165
Create a Named Selection that Selects All Faces at a Specified Location	165
Rescope a Solved Result Based on the Active Node or Element Selection	166
Scope a Boundary Condition to a Named Selection	166
Add a Joint Based on Proximity of Two Named Selections	167
Print Selected Element Faces	168
Get Normal of a Face	169
Create a Selection Based on the Location of Nodes in Y	169
Create Aligned Coordinate Systems in a Motor	170
Script Examples for Interacting with Tree Objects	173
Delete an Object	174
Refresh the Tree	174
Clear the Mesh	174
Get All Visible Properties for a Tree Object	174
Parametrize a Property for a Tree Object	174
Create Construction Lines from Cylindrical Faces	175
Update Geometries for all Construction Lines	175
Create Material Assignment from Body Materials	176
Count the Number of Contacts	176
Suppress Duplicate Contacts	177
Verify Contact Size	178
Set Pinball to 5mm for all Frictionless Contacts	178
Use a Named Selection as Scoping of a Load or Support	178
Suppress Bodies Contained in a Given Named Selection	179
Rename a Named Selection Based on Scoping	179
Modify the Scoping on a Group of Objects	180
Change Tabular Data Values of a Standard Load or Support	180
Duplicate an Harmonic Result Object	181
Retrieve Object Details Using SolverData APIs	181
Evaluate Spring Reaction Forces	181
Export a Result Object to an STL File	182

Export Result Images to Files	182
Tag and Group Result Objects Based on Scoping and Load Steps	182
Work with Solution Combinations	184
Create a Pressure Load	184
Create a Convection Load	185
Create Node Merge Object at a Symmetry Plane	185
Access Contour Results for an Evaluated Result	186
Write Contour Results to a Text File	187
Access Contour Results at Individual Nodes/Elements	187
Set Arbitrary Coordinate System Properties	188
Transform Coordinate Systems (with Math)	188
Select Objects By Name	189
Export Figures	190
Script Examples for Interacting with the Mechanical Session	193
Remesh a Model Multiple Times and Track Metrics	193
Perform Solution While Specifying Solution Handler and the Number of Cores	194
Scan Results, Suppress Any with Invalid Display Times, and Evaluate	194
Check Version	194
Check Operating Environment	195
Search for Keyword and Export	195
Modify Export Setting	195
Pan the Camera	195
Functions to Draw	196
Export All Result Animations	198
Get User Files Directory	199
Display an Arrow at the Centroid of Selected Faces	199
Compute Shortest Distance Between Two Faces	200
Display Extensions Loaded in Mechanical	203
Script Examples for Interacting with Results	205
Retrieve Stress Results	205
Create Probe Principal Stresses from a Node Selection	205
Finding Hot Spots	206
Examples Using the Python Code Object	211
After Object Changed Filtering	211
Send a Warning Message	212
Export a Boundary Condition	212
Update Following Mesh Generation	213
Check Property Values	214
Change Material Property	214
Print Message to ACT Log	215
Property Provider Example	215
Parameterize Coordinate System Transformation	221
Specify and Parameterize Bolt Pretension Loads	222
End-to-End Analysis Examples	227
Static Structural Analysis	228
Static Structural Spatially Varying Load Analysis	230
Transient Structural Analysis	232
Static Structural General Joint Analysis	236
Static Structural Universal Joint Analysis	238
Transient Structural Cylindrical Joint Analysis	240
Symmetric Symmetry Analysis	243

Linear Periodic Symmetry Analysis	246
Cyclic Symmetry Analysis	249
Steady-State Thermal Analysis	254
Transient Thermal Analysis	256
Coupled Field Static Analysis	258
Coupled Field Harmonic Analysis	260
Coupled Field Transient Analysis	263
Coupled Field Modal Analysis	265
Fracture Analysis: Semi-Elliptical Crack	267
Fracture Analysis: SMART Crack Growth	270
Fracture Analysis: Contact Debonding	276
Fracture Analysis: Interface Delamination	279
Harmonic Acoustic Analysis	282
Modal Acoustic Analysis	285
Structural Optimization (Density Based) Analysis	287
Structural Optimization (Level Set Based) Analysis	291
Structural Optimization Lattice Analysis	294
Structural Optimization Shape Optimization Analysis	297
Rigid Dynamics: Contact Specification	300
Rigid Dynamics: Flexible Part Specification	302
Rigid Dynamics: General Analysis	305
Rigid Dynamics: Joint Specification	307
Rigid Dynamics: Variable Load Specification	309
Steady State Electric Conduction Analysis	312
Steady-State Thermal-Electric Conduction Analysis	315
Magnetostatic Analysis	317
Post Processing Options	320
Post Processing User Defined Results	322
Random Vibration Analysis	324
Static Structural Analysis using External Model	330

Scripting Quick Start

Scripting refers to the use of a programming language to interact with and modify a software product. Scripting can also be used to automate routine tasks. In Ansys Mechanical, you can use Ansys ACT and Mechanical Python APIs (Application Programming Interfaces).

[Scripting Introduction](#)

[Key Usage Concepts](#)

[Threading](#)

[Additional Resources](#)

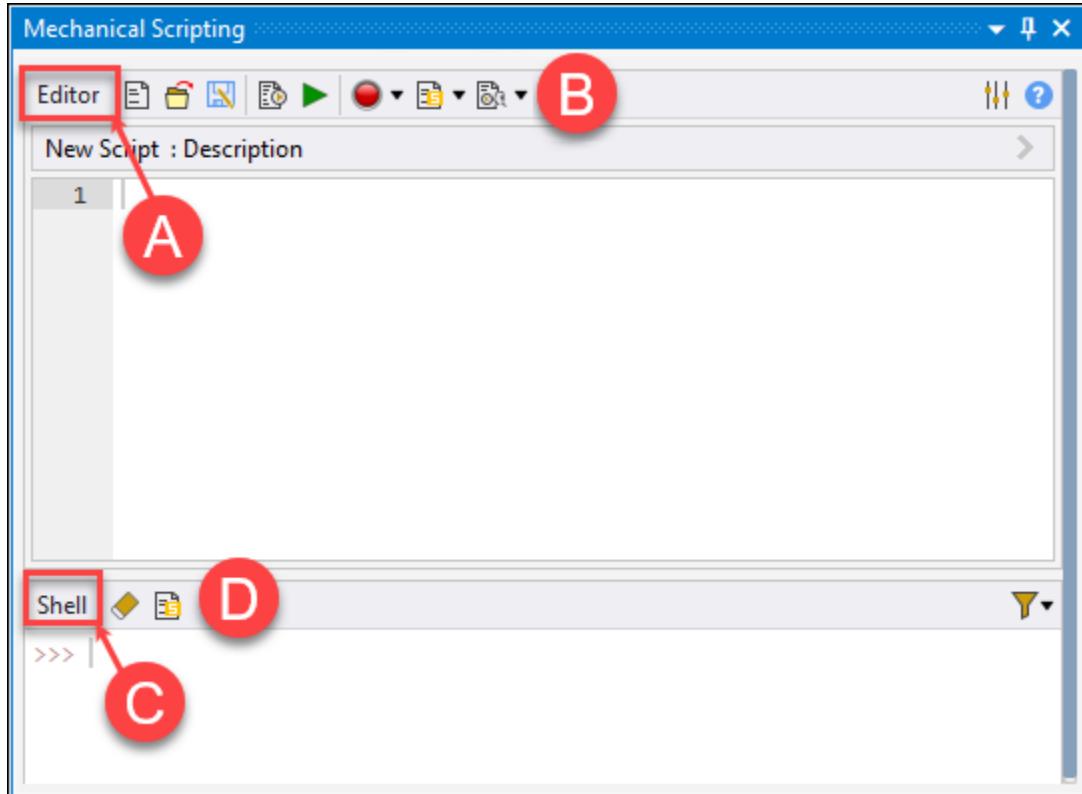
Scripting Introduction

This section provides introductory information about scripting in Mechanical:

- Mechanical Scripting View
- Autocompletion
- Snippets
- Keyboard Shortcuts
- Recording APIs
- Debugging Scripts and Extensions
- Scope Selection for ACT Extensions

Mechanical Scripting View

The Mechanical **Scripting** view provides a multi-line editor and shell for APIs so that you can develop and debug scripts. You can show and hide the **Scripting** view from the ribbon's **Automation** tab. In the **Mechanical** group, clicking **Scripting** switches between showing and hiding this view.



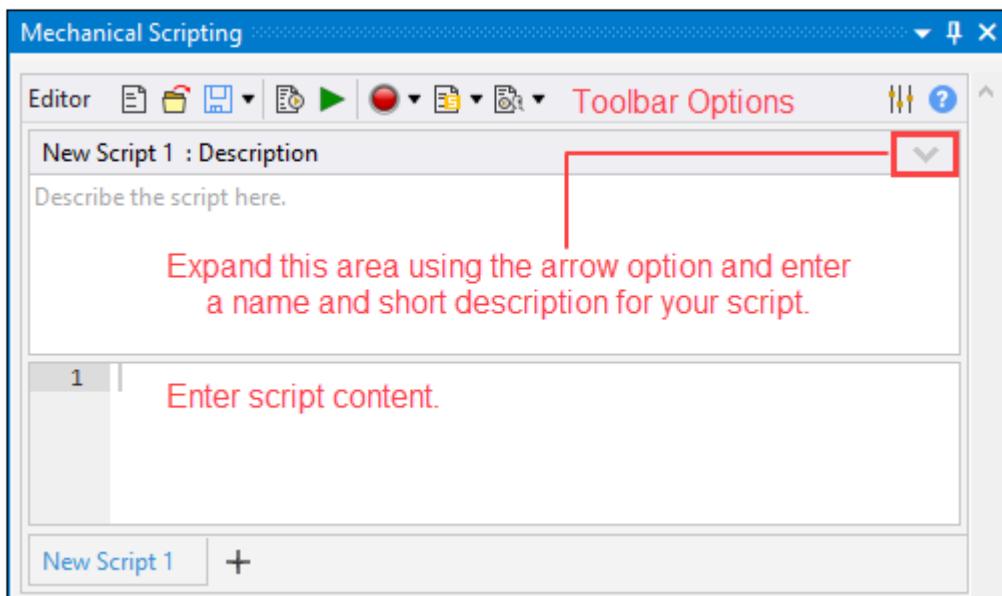
Callout	Name	Description
A	Editor	Work on long scripts for your workflows.
B	Editor Toolbar	Interact with scripts by performing actions such as opening scripts from disk, promoting scripts to buttons, and more.
C	Shell	Work with shorter commands to build your scripts.
D	Shell Toolbar	Interface with the shell by performing actions such as clearing the shell and opening the snippet inserter.

Note:

- The **Scripting** view initially opens in a locked position to the right of the graphics view. You can click the header bar and drag this view to anywhere in Mechanical, dropping it into a new locked position. By double-clicking the header bar, you can toggle between the locked position and a floating window. You can resize and move the floating window as needed.
- If you have ACT extensions loaded and debug mode is enabled, below the **Shell** area, a tab displays for each loaded extension so that you can set the scope. For more information, see [Scope Selection for ACT Extensions \(p. 29\)](#).
- You can revert to the **ACT Console** by changing the scripting view preference under **File > Options > Mechanical > UI Options > New Scripting UI**. Mechanical must be restarted to see the scripting view change.

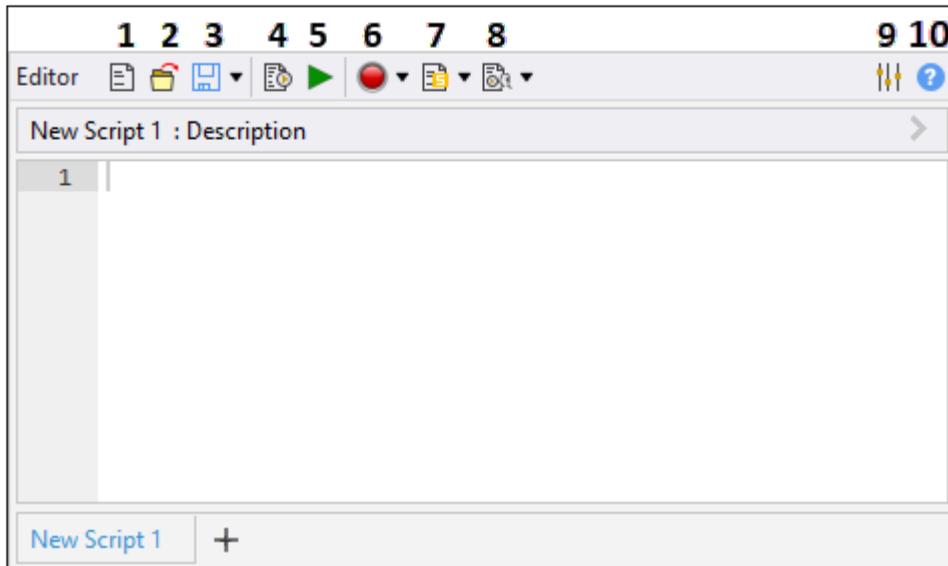
Working with the Editor

The **Editor** appears in the top panel of the Mechanical **Scripting** view.



Toolbar

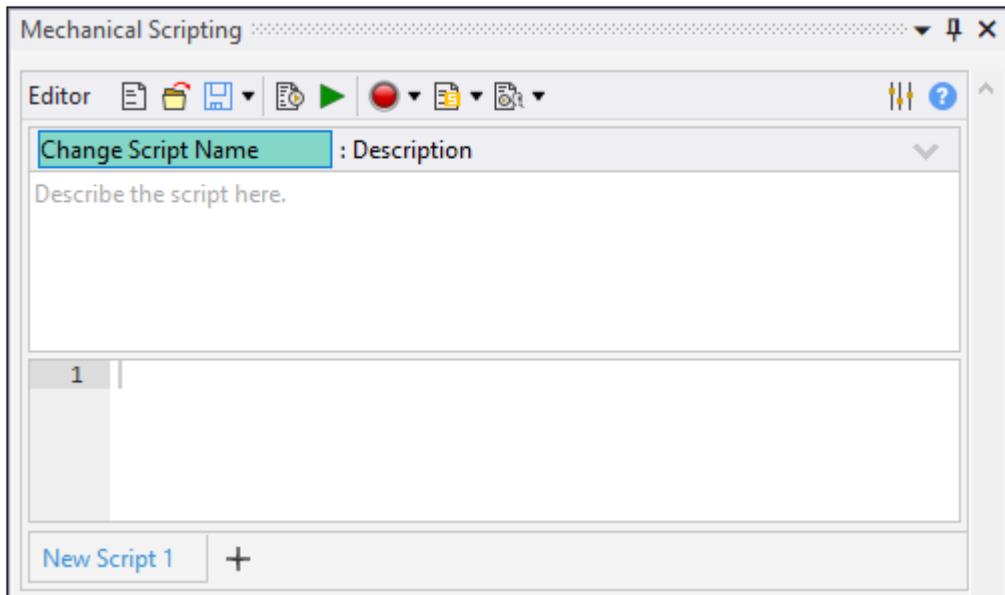
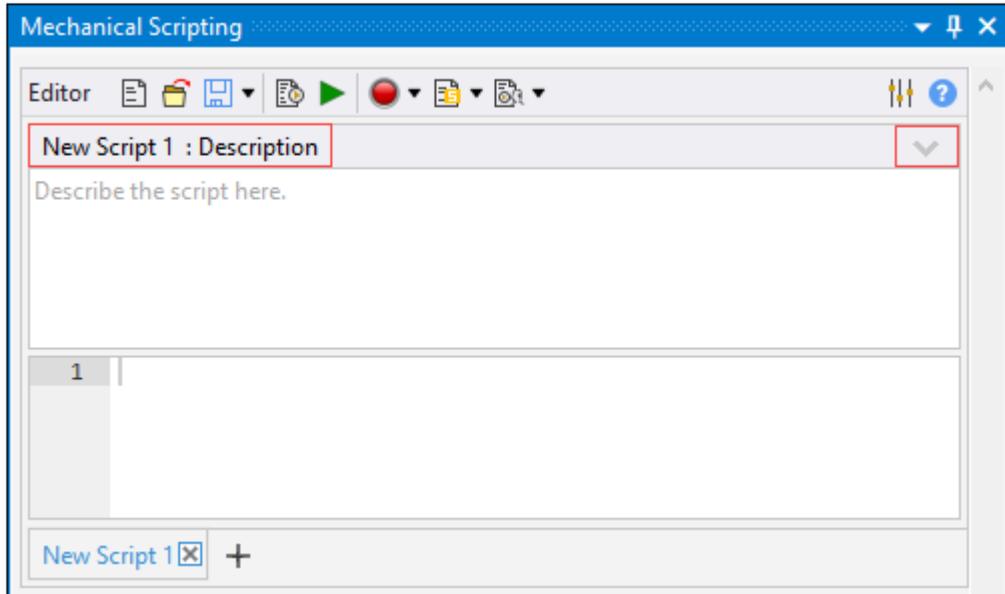
The toolbar for the **Editor** provides several buttons:



1. **New Script:** Clears the name, description, and script boxes, providing you with a new script template in which to work.
2. **Open Script:** Opens a script, which is any Python file (`*.py`), from disk.
3. **Save Script (Ctrl+S)/Save Script As:** Save your script to disk.
4. **Run Script:** Executes the script contained in the pane.
5. **Start Debugger:** This runs the script in debug mode. See the [Debugging \(p. 24\)](#) section for more information.
6. **Start Recording:** Automatically create APIs by recording the actions that you take. For example, inserting a load or a result. This option also includes a drop-down menu where you can select to active the Camera and/or Graphics filters. See descriptions for these options in the [Default Settings \(p. 8\)](#) topic below.
7. **Insert Snippet (Ctrl+I):** Open the [snippet inserter \(p. 16\)](#). This option includes a drop-down menu for the **Promote Script to Snippet** and **Open Snippet Editor** options. Both of the options open the [Snippet Editor \(p. 17\)](#). For the promotion option, certain fields are automatically completed in the Editor.
8. **Show Button Editor:** Opens the **Button Editor**. This option includes a drop-down menu with the option **Promote Script to Button** that opens the **Button Editor** with certain fields automatically completed using the information from the script in the **Editor**. For more information, see [Creating User-Defined Buttons](#) in the *Mechanical User's Guide*.
9. **Default Settings:** Display the [Default Settings \(p. 8\)](#) dialog (see below). This dialog enables you to specify the default settings for recording and what is displayed of the Shell pane.
10. **Help:** Displays a list of keyboard shortcuts.

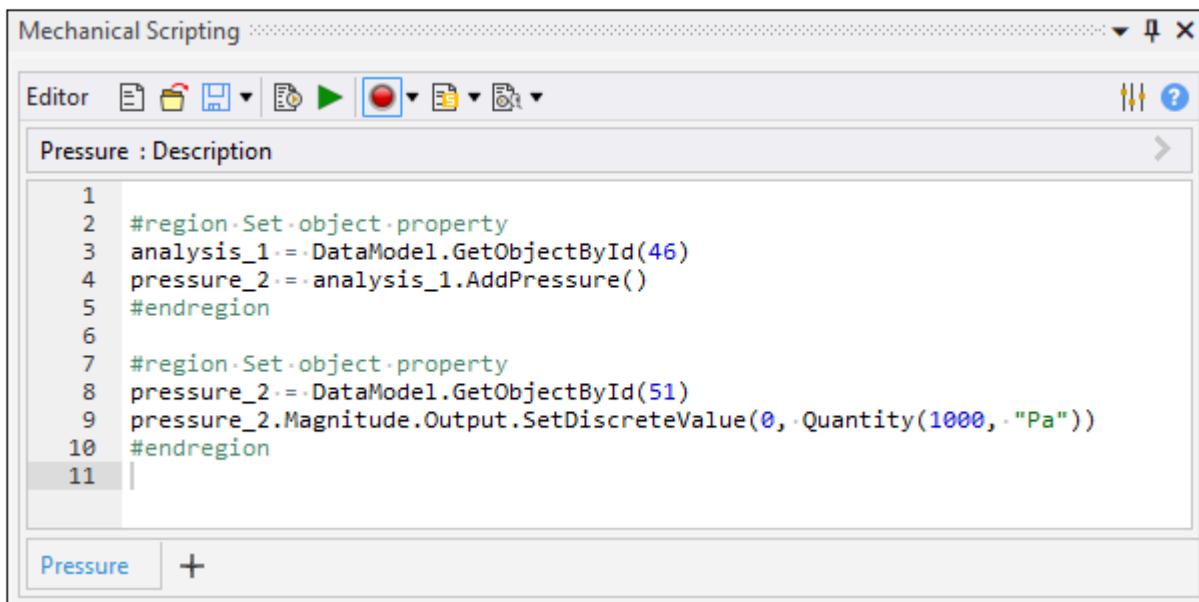
Name and Description Area

As illustrated, there is a field for your script's name. The default is simply "New Script #." Select the field to specify a name. Using the down arrow option, you can display a pane in which you add a description for the script.



Content Area

The content area is where you write scripts, such as the example Pressure illustrated below.

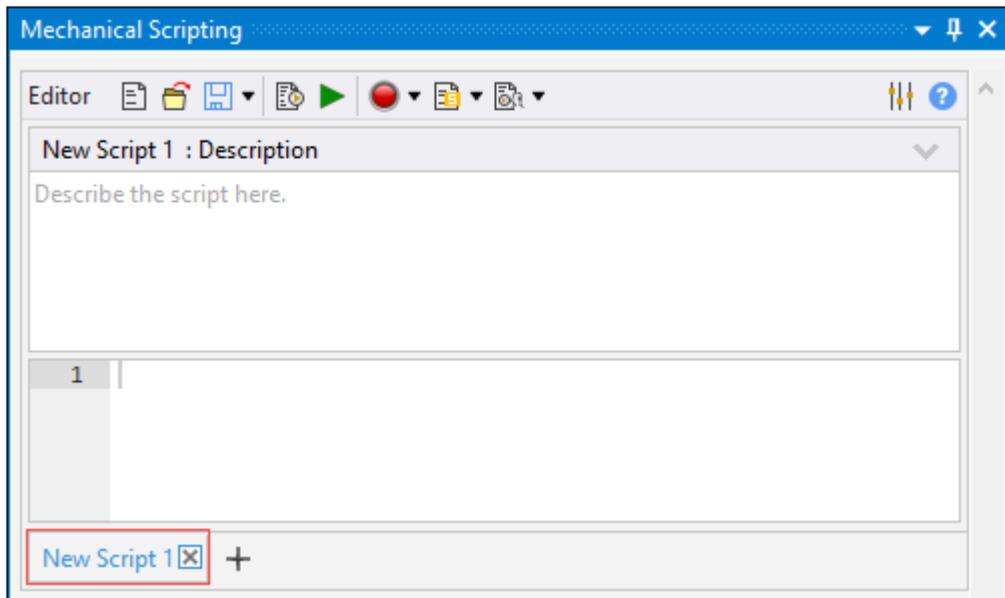


The content area provides [autocompletion](#) (p. 10) and the following shortcuts:

- **Ctrl + f:** Opens the find and replace dialog box.
- **Ctrl + F5:** Executes the script.
- **Ctrl + i:** Opens the [snippet inserter](#) (p. 16).

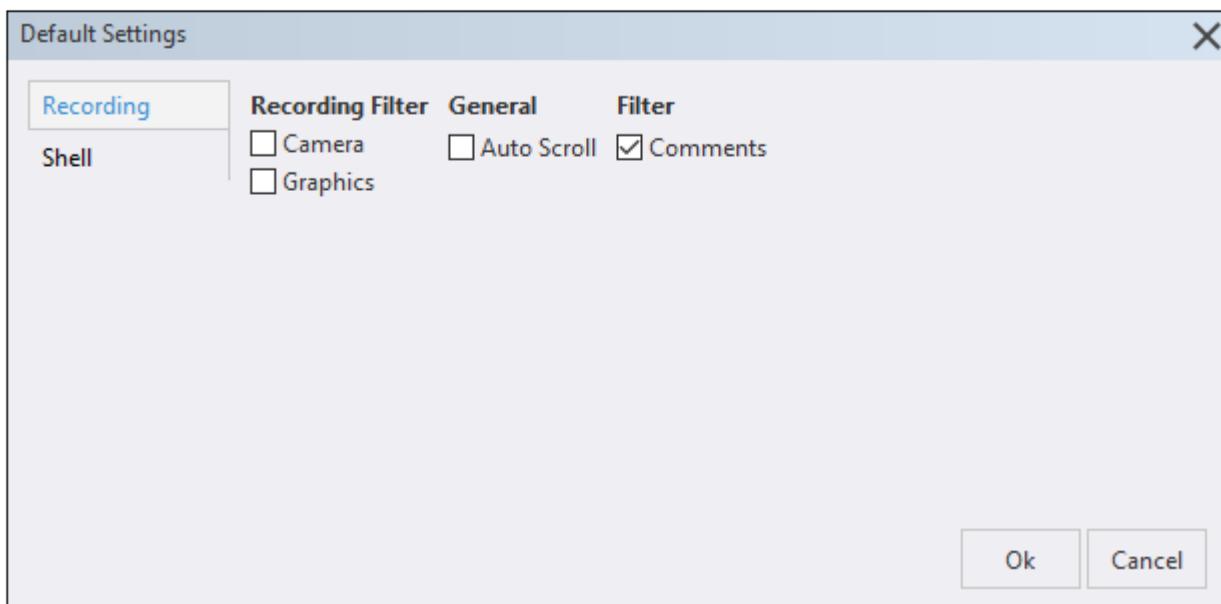
Script Tabs

A script "Tab" displays at the bottom of the pane. You can now work on multiple scripts at the same time using tabs. You add new tabs using the **+** button and delete tabs using the **x** button that displays when you hover over a tab. When you open a new script, it will now open in a new tab. When you have multiple tabs, you can switch between scripts.



Default Settings

Selecting the **Default Options** button displays the following dialog.



Settings include:

- **Camera:** Selecting this option enables you to capture all movements (pan/rotate) made on the model in the **Geometry** window are recorded.
- **Graphics:** Selecting this option enables you to capture certain Graphical Toolbar options and display features while recording. Examples include turning the ruler on and off, creating Section Planes, and selecting the Thick Shells and Beams options.
- **Auto Scroll:** As entries or recordings are made in the script pane, the pane automatically scrolls as new content is added.

- **Comments:** Selected by default, unchecking this option turns off the automatic comments (green text) that are added for each API entry or recording.

See the [Working with the Shell \(p. 9\)](#) section for a description of the Shell filters provided via the dialog. These same filters are available from the Shell pane **Shell Content Filter** option.

Working with the Shell

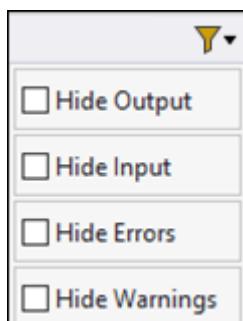
The **Shell** appears in the lower panel of the Mechanical **Scripting** view.



Toolbar

The toolbar for the **Shell** provides these buttons:

1. **Clear Contents:** Clears the contents of the **Shell**. This can also be done by executing the command `clear` inside the **Shell**.
2. **Insert Snippet:** Opens the [snippet inserter \(p. 16\)](#) in context of the **Shell**.
3. **Shell Preferences:** Displays a drop-down menu for indicating whether to hide the output, input, errors, and warnings. All check boxes are cleared by default so that the output, input, errors, and warnings are all shown.



4. **Content area:** Area in which you enter and execute single-line or multi-line commands. This area provides [autocompletion \(p. 10\)](#).

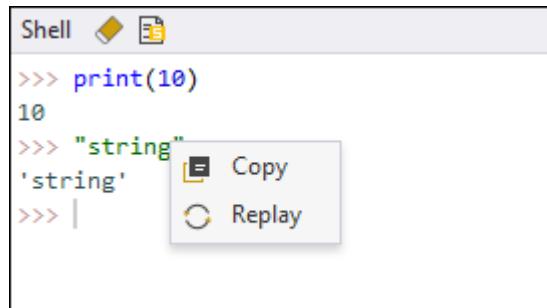
Color Coding for Output in the Shell

In the output, the color of the text distinguishes output from errors and warnings.

Text Color	Item
Gray	Output
Red	Error
Yellow	Warning

Context Menu

There is a context menu associated with the commands that have been executed. To display this context menu, place the mouse cursor over the executed command and right-click.



- **Copy:** Copies the command to the clipboard.
- **Replay:** Re-inserts the executed command in the content area for execution.

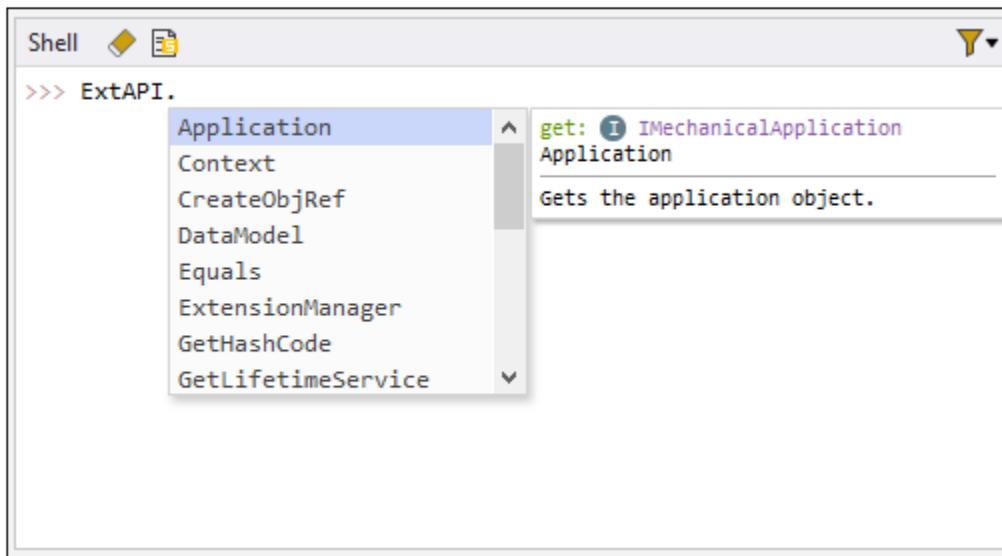
Shortcuts:

The **Shell** provides these shortcuts:

- **Ctrl + ↑:** Takes you to the previously executed command.
- **Ctrl + ↓:** Takes you to the next executed command.
- **Enter:** Executes the command.
- **Shift + Enter:** Inserts a new line.

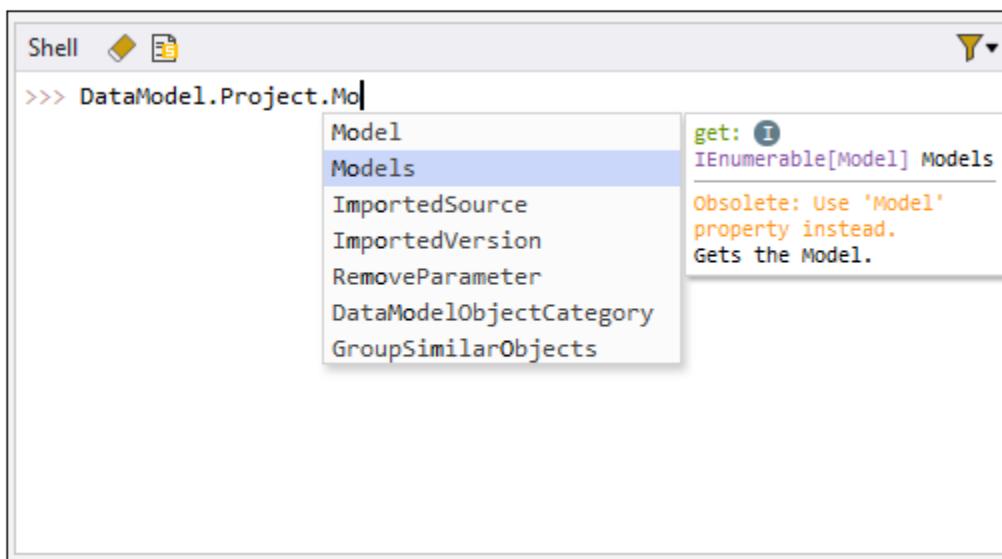
Autocompletion

The text editor in the **Editor**, **Snippet Editor** (p. 17), and the input field in the **Shell** all provide autocompletion.



Autocompletion Tooltips

When you place the mouse cursor over any property or method in the list of suggestions provided, the tooltip displays information about this item. The following image shows the tooltip for the property **Models**.



Tooltips use color-coding to indicate the syntax:

Color	Syntax
Green	Accessibility
Purple	Type
Orange	Warning
Blue	Argument

Properties

General formatting for properties follow:

- **get/set mode:** *ReturnType PropertyName*
- Description of the property
- **Returns:** Description of what is returned (if any)
- **Remarks:** Additional information (if any)
- **Example:** Sample entry (if any)

Methods

General formatting for methods follow:

- *ReturnType MethodName (Argument Type Argument Name)*
- **Argument Name:** Description of the argument
- **Returns:** Description of what is returned (if any)
- **Remarks:** Additional information (if any)
- **Example:** Sample entry (if any)

Additional Information

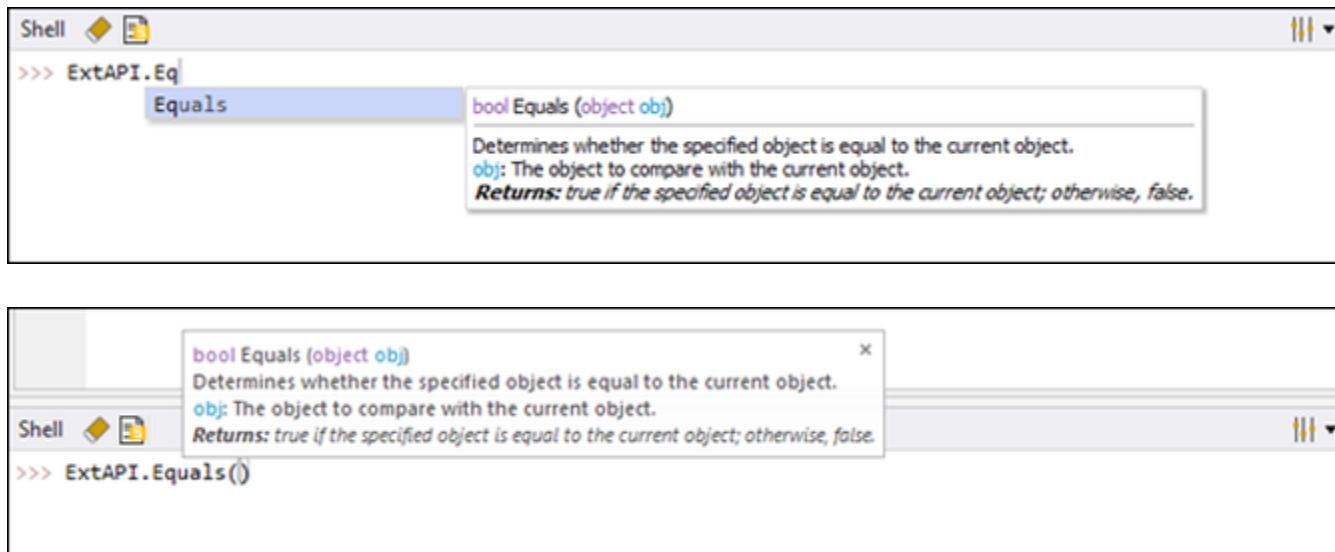
Tooltips can also provide:

- .NET properties where applicable
- **Warning messages** when special attention is needed
- **Prototype information** for methods when cursor is inside brackets and indexers when cursor is inside square brackets
- Overloaded methods, including the following details:
 - Number of overloaded methods
 - Prototypes for overloaded methods (accessed by pressing the arrows in the prototype tooltip)
 - Members for overloaded methods

The tooltip for an overloaded method is a list of all members from all overloaded methods.

Tooltip Examples

The following images show two different tooltip examples for the method `Equals` in two different stages of using autocomplete:



Keyboard Shortcuts:

The tooltip provides these keyboard shortcuts:

- **Enter:** Inserts the suggestion.
- **Esc:** Closes autocomplete tooltip.
- **Ctl + Space:** Forces autocomplete to reopen.

Snippets

Snippets are code or code templates that you can quickly and easily insert in the command line, saving you from repetitive typing. As you write scripts, you can insert any of the snippets that Ansys supplies. Additionally, you can begin building your own library of snippets to either supplement or replace supplied snippets with your own custom snippets.

For more information, see:

- [Supplied Snippets](#)
- [Snippet Usage Example](#)
- [Snippet Inserter](#)
- [Snippet Creation and Management](#)

Supplied Snippets

Descriptions follow of all supplied snippets:

The snippet **ExtAPI** inserts **ExtAPI.** in the command line, providing you with immediate access to the ACT API. From the autocomplete options in the tooltip, you can then begin selecting members to build your command.

The snippet **Snippet Template** provides sample code for swapping two variables. The comments explain how a snippet can contain editable fields, which are automatically selected for modification when the snippet is inserted. When a field is selected, you can type a new value to override the default value or let the default value remain. Pressing the **Tab** key moves the cursor to the next editable field.

The folder **Helpers** provides snippets for frequently used commands:

- **Project.** Inserts `project = DataModel.Project`, providing access to the project, which is the top level of the hierarchy in the Mechanical tree. To interact with first-level objects in the tree, you would then type a period and use the list of suggestion to select the attribute `Model` and then the attribute for the first-level object. For example, these command line entries provide access to the first-level tree objects `Connections` and `Named Selections`:

```
- project=DataModel.Project.Model.Connections  
- project=DataModel.Project.Model.NamedSelections
```

The snippets **Mesh** and **Geometry** provide examples of easier methods for accessing first-level objects.

- **Mesh.** Inserts `mesh = Model.Mesh`, providing access to the object `Mesh`.
- **Geometry.** Inserts `geometry = Model.Geometry`, providing access to the object `Geometry`.
- **Analysis.** Inserts `analysis = Model.Analyses[0]`, providing access to the first analysis of the model. As indicated earlier, Python starts counting at 0 rather than 1.
- **ObjectsByName.** Inserts `solution = DataModel.GetObjectsByName("Solution")`, providing access to a list of all solutions. To apply this function with pressure, you use `solution = DataModel.GetObjectsByName("pressure")`.
- **PathToFirstActiveObject.** Inserts `path_to_object = Tree.GetPathToFirstActiveObject()`, providing access to the full path that must be typed to go to the first object that is selected in the tree.
- **ObjectsByType.** Inserts `coordinate_system_list = DataModel.GetObjectsByType(DataModelObjectCategory.CoordinateSystem)`, where `DataModelObjectCategory` is the enumeration containing all types. To apply this function with pressure, you use `pressure_list = DataModel.GetObjectsByType(DataModelObjectCategory.Pressure)`.
- **Active Objects.** Inserts `active_objects = Tree.ActiveObjects`, providing a list of all selected objects in the tree.
- **FirstActiveObject.** Inserts `first_active_object = Tree.FirstActiveObject`, providing access to the first of all selected objects.

- **Quantity.** Inserts `Quantity("1 [mm]"`, providing a command method in which you can declare values. Three examples follow:

```
pres = DataModel.Project.Model.Analyses[0].AddPressure()

pres.Magnitude.Inputs[0].DiscreteValues = [Quantity('0 [sec]'), Quantity('1 [sec]'), Quantity('2 [sec]')]

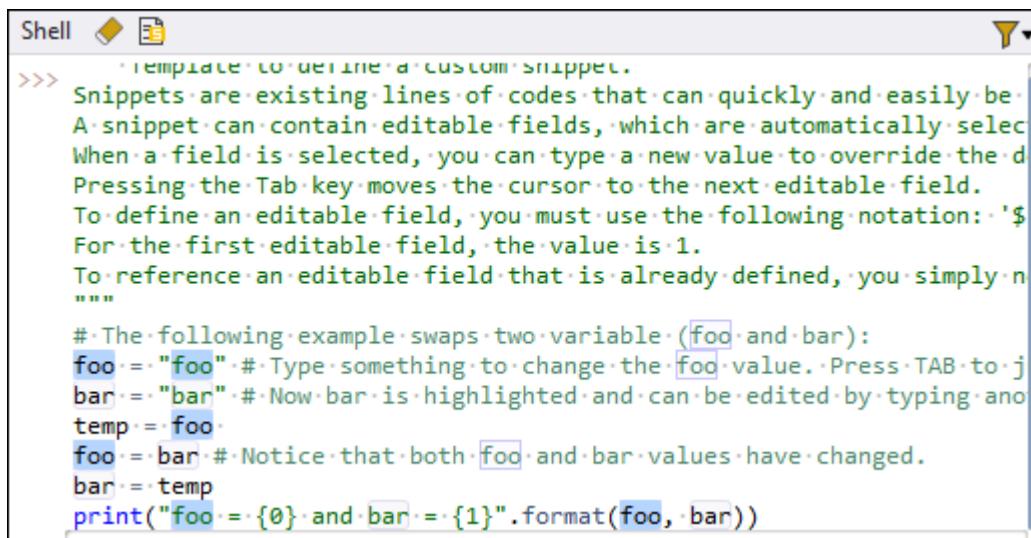
pres.Magnitude.Output.DiscreteValues = [Quantity('0 [Pa]'), Quantity('50 [Pa]'), Quantity('100 [Pa]')]
```

- **Selection Manager.** Inserts `selection_manager = ExtAPI.SelectionManager`, providing access to the Selection Manager. This object contains properties and functions related to graphical selection within Mechanical, such as getting information about the current selection or modifying the selection.
- **Model View Manager.** Inserts `model_view_manager Graphics.ModelViewManager`, providing access to the Model View Manager. This object provides functionality to control managed graphical views.

Lastly, the folder **Examples** provides snippets that you can use as templates. For example, the snippet **Add Pressure** shows how to create a pressure on the first face of the first body of the first part.

Snippet Usage Example

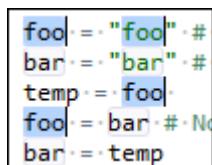
When a snippet is inserted, you can easily see all editable fields in the command line.



The screenshot shows the ANSYS Mechanical software interface with the 'Shell' tab selected. In the command line area, there is a snippet template. The template starts with a comment: `>>> # Replace to define a custom snippet.`. It then describes snippets and how they work. It includes an example of swapping variable values:

```
>>> Snippets are existing lines of code that can quickly and easily be used. A snippet can contain editable fields, which are automatically selected when a field is selected, you can type a new value to override the default. Pressing the Tab key moves the cursor to the next editable field. To define an editable field, you must use the following notation: '$'. For the first editable field, the value is 1. To reference an editable field that is already defined, you simply name it.
# The following example swaps two variables (foo and bar):
foo = "foo" # Type something to change the foo value. Press TAB to jump to the next field.
bar = "bar" # Now bar is highlighted and can be edited by typing another value.
temp = foo
foo = bar # Notice that both foo and bar values have changed.
bar = temp
print("foo = {0} and bar = {1}".format(foo, bar))
```

For example, in **Snippet Template**, the first editable field (`foo`) is highlighted.



```
foo = "foo" # Type something to change the foo value. Press TAB to jump to the next field.
bar = "bar" # Now bar is highlighted and can be edited by typing another value.
temp = foo
foo = bar # Notice that both foo and bar values have changed.
bar = temp
```

Typing something changes the `foo` value to whatever you type (`value1`). Notice that both `foo` values change to `value1` in one operation.

```

value1 = "value1"
bar = "bar" # Now
temp = value1
value1 = bar # Not
bar = temp

```

Pressing the **Tab** key moves the cursor to the next editable field, causing **bar** to be highlighted.

```

value1 = "value1"
bar = "bar" # Now
temp = value1
value1 = bar # Not
bar = temp

```

Typing something changes the **bar** value to whatever you type (**value2**). Notice that both **bar** values change to **value2** in one operation.

```

value1 = "value1" #
value2 = "value2" #
temp = value1
value1 = value2 # N
value2 = temp

```

Pressing the **Tab** key again finalizes the code.

To define an editable field, you must use the following notation: `#{#:default_value}`, where `#` is the index of the field. For the first editable field, the value is `1`. To reference an editable field that is already defined, you simply need to specify its index (`#{#}`) as shown in the following figure for the snippet **Geometry**.

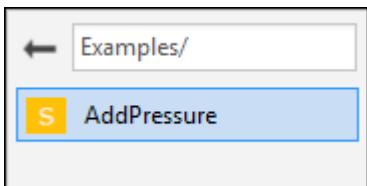


Snippet Inserter

The snippet inserter can be accessed in the context of the **Editor** or the **Shell**. You can use the buttons on the toolbar or the keyboard shortcut **Ctrl + i** to open the snippet inserter. It will open wherever your cursor is in the script. Once the snippet inserter is opened, you can use the mouse or the keyboard to interact with it.

Mouse Interaction

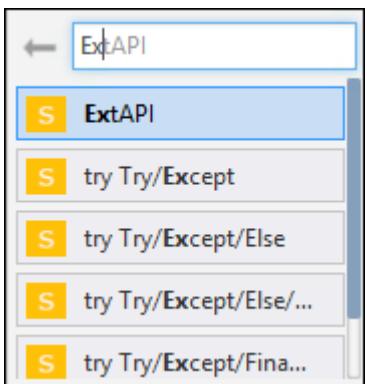
Using the mouse, you can simply click snippets to insert them into the **Editor**. If you click a snippet folder, it opens so that you can see the snippets inside it. As you browse folders, the text in the search box displays your current file path:



Clicking the back button for the search box returns you to your last location.

Keyboard Interaction

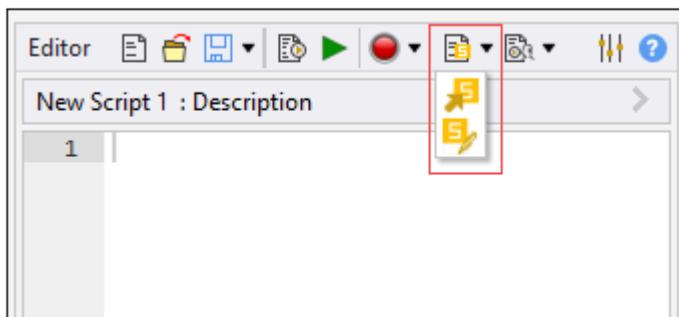
You can use the search box to search or browse snippets. As seen in the following figure, if you type **Ex**, the snippet inserter displays all snippets and snippet folders that include **Ex** in their names. From there, you can use the **Tab** or **Enter** key to select a snippet or a snippet folder.

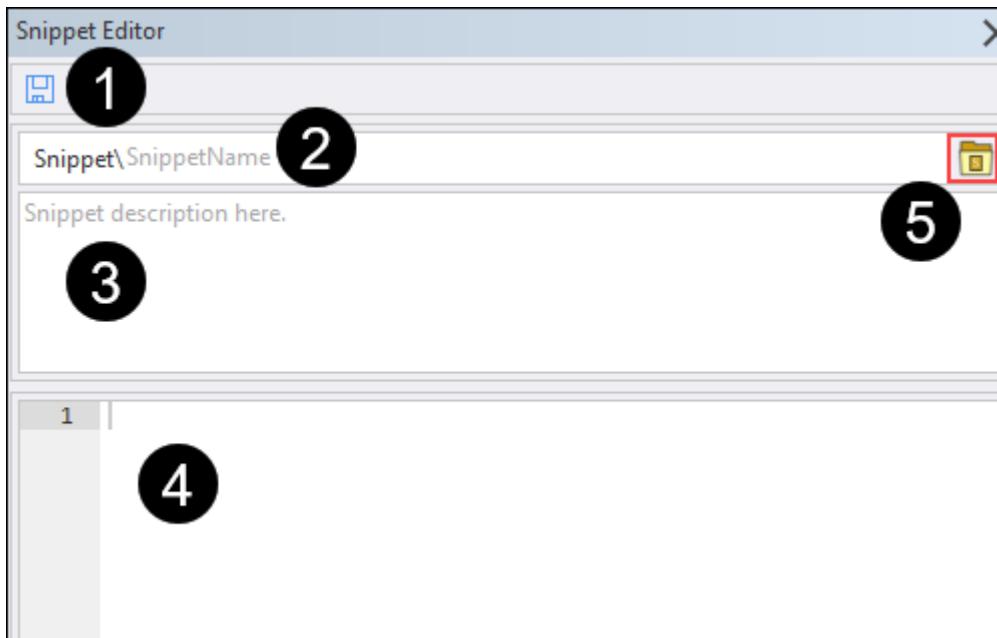


Deleting the text in the search box or using the keyboard shortcut **Alt + ←** returns you to your last location.

Snippet Creation and Management

To create snippets, you use the **Snippet Editor**, which you access from the toolbar for the **Editor**:





1. **Save button:** Saves the snippet to the path specified.
2. **Name input field:** Specifies the name or both the folder and name to which to save the snippet. Examples follow:
 - To save a snippet named **Quick Add** to the root folder for snippets, the name input field would look like this:

Snippet\Quick Add

Note:

Root folder refers to the place where you start out when browsing snippets. For reference, the snippet **ExtAPI** is in the root folder.

-
- To save this same snippet to the folder **Examples**, the name input field would look like this:

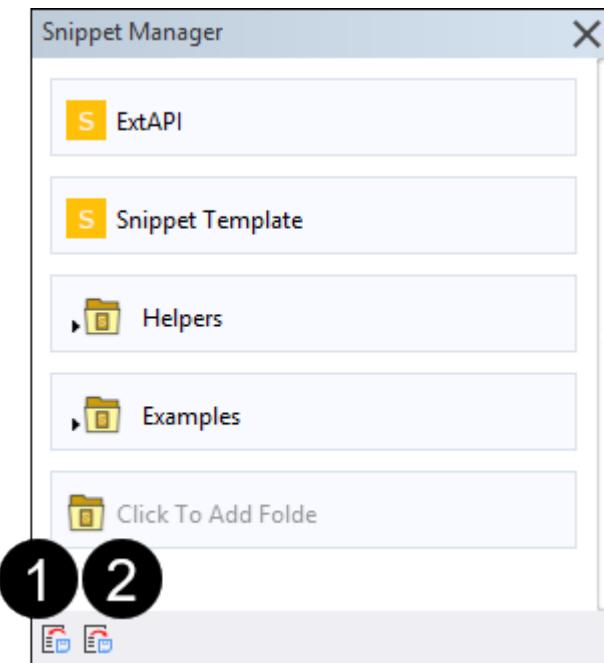
Snippet\Examples\Quick Add

Note:

When you click an existing folder, the **Snippet Manager** inserts the path to this folder in the name input field so that the snippet will be saved to this folder. You can also manually type the path to the folder.

-
3. **Description:** Describes what your snippet does.

4. **Content area:** Where you write the snippet that is to be inserted when the snippet inserter is used.
5. **Snippet Manager:** Opens a view where you can manage your snippets.



Descriptions follow for the two buttons in the lower left corner of the **Snippet Manager**:

- a. **Import snippet collection:** Opens a dialog box for you to select the snippet collection file. The **Snippet Manager** supports XML and JSON files. The XML files can be files that were exported using the **ACT Console**. The JSON files are snippet collections that are imported using the **Snippet Manager**. Importing snippet collection files will add the snippets that they contain to your existing snippets.
- b. **Export snippet collection:** Exports all snippets in the **Snippet Manager** to a JSON file.

Note:

The **Snippet Manager** displays the **Click To Add Folder** option as the last node in every folder. You can click this option to create a new folder in the current folder.

The other options available to you in the **Snippet Manager** depend on whether you place the mouse cursor over a snippet or a snippet folder.

When you place the mouse cursor over a snippet, buttons are available for either editing or deleting the snippet:

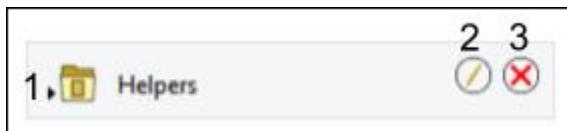


1. **Edit snippet:** Opens the snippet in the **Snippet Editor** so that you can view and make changes to the snippet.
 2. **Delete snippet:** Deletes the snippet.
-

Caution:

Deleting a snippet from the **Snippet Manager** also removes it from the snippet inserter.

When you place the mouse cursor over a folder, buttons are available for browsing the folder and either editing or deleting the folder:



1. **Browse folder:** Expand the folder to browse the snippet collection.
 2. **Edit folder:** Makes the folder name editable so that you can change it.
 3. **Delete folder:** Deletes the folder.
-

Caution:

Deleting a folder deletes all of its contents.

Keyboard Shortcuts

The following topics summarize the keyboard shortcuts that are available.

[Editor and Shell Keyboard Shortcuts](#)

[Text Editor Keyboard Shortcuts](#)

Editor and Shell Keyboard Shortcuts

These keyboard shortcuts are available in the **Editor**, **Shell**, or both areas.

Editor

The **Editor** supports **Ctrl + F5** as a shortcut for executing the script.

Shell

The **Shell** supports the following shortcuts:

- **Enter:** Executes the command

- **Shift + Enter:** Inserts a new line.
- **Ctrl + ↑:** Takes you to the previously executed command.
- **Ctrl + ↓:** Takes you to the next executed command.
- **Esc:** Closes autocompletion tooltips.

Editor and Shell

Both the **Editor** and **Shell** support the following shortcuts:

- **Ctl + i:** Opens the snippet inserter.
- **Ctl + Space:** Forces autocompletion to reopen.

Text Editor Keyboard Shortcuts

The following tables list keyboard shortcuts for the text editor.

[Line Operation Shortcuts](#)

[Selection Shortcuts](#)

[Multi-Cursor Shortcuts](#)

[Go-To Shortcuts](#)

[Folding Shortcuts](#)

[Other Shortcuts](#)

Line Operation Shortcuts

Key Combination	Action
Ctrl + D	Remove line
Alt + Shift + ↓	Copy lines down
Alt + Shift + ↑	Copy lines up
Alt + ↓	Move lines down
Alt + ↑	Move lines up
Alt + Backspace	Remove to line end
Alt + Delete	Remove to line start
Ctrl + Delete	Remove word left
Ctrl + Backspace	Remove word right

Selection Shortcuts

Key Combination	Action
Ctrl + A	
Shift + ←	Select all
Shift + →	Select left
Ctrl + Shift + ←	Select right
Ctrl + Shift + →	Select word left
Shift + Home	Select word right
Shift + End	Select line start
Alt + Shift + →	Select line end
Alt + Shift + ←	Select to line end
Shift + ↑	Select to line start
Shift + ↓	Select up
Shift + Page Up	Select down
Shift + Page Down	Select page up
Ctrl + Shift + Home	Select page down
Ctrl + Shift + End	Select to start
Ctrl + Shift + D	Select to end
Ctrl + Shift + P	Duplicate selection

Multi-Cursor Shortcuts

Key Combination	Action
Ctrl + Alt + ↑	Add multi-cursor above
Ctrl + Alt + ↓	Add multi-cursor below
Ctrl + Alt + →	Add next occurrence to multi-selection
Ctrl + Alt + ←	Add previous occurrence to multi-selection
Ctrl + Alt + Shift + ↑	Move multi-cursor from current line to the line above
Ctrl + Alt + Shift + ↓	Move multi-cursor from current line to the line below
Ctrl + Alt + Shift + →	Remove current occurrence from multi-selection and move to next
Ctrl + Alt + Shift + ←	Remove current occurrence from multi-selection and move to previous
Ctrl + Shift + L	Select all from multi-selection

Go-To Shortcuts

Key Combination	Action
Page Up	Go to page up
Page Down	Go to page down
Ctrl + Home	Go to start
Ctrl + End	Go to end
Ctrl + L	Go to line
Ctrl + P	Go to matching bracket

Folding Shortcuts

Key Combination	Action
Alt + L, Ctrl + F1	Fold selection
Alt + Shift + L, Ctrl + Shift + F1	Unfold

Other Shortcuts

Key Combination	Action
Tab	Indent
Shift + Tab	Outdent
Ctrl + Z	Undo
Ctrl + Shift + Y, Ctrl + Y	Redo
Ctrl + T	Transpose letters
Ctrl + Shift + U	Change to lower case
Ctrl + U	Change to upper case
Insert	Overwrite

Recording APIs

Selecting the Recording option on the toolbar enables you to automatically generate ("record") application supported APIs based on the actions that you make in the application, such as contact surfaces, loading, and the definition of results. For the animated example shown here, a **Fixed Support** and a **Pressure** were added to the **Outline**. The recording feature captured these actions. This feature is a convenient tool for API development. See the [Default Settings \(p. 8\)](#) topic in the *Working with the Editor* section for the filters available for this feature.

Points to Remember

- When you record and save API content for moving the model around in the **Geometry** window, via the Camera Default Setting, those APIs are independent of the geometry. If you save movement APIs (pan, rotate, etc.), they will execute of any model in the application.
- For saved script files, the order in which you record your actions and create APIs has consequence. You need to ensure that all APIs support any redefinition or changes as you progress through a recording.

Note:

This feature does not support every available API. Further development in this regard as well as the improvement of the feature itself is ongoing and required.

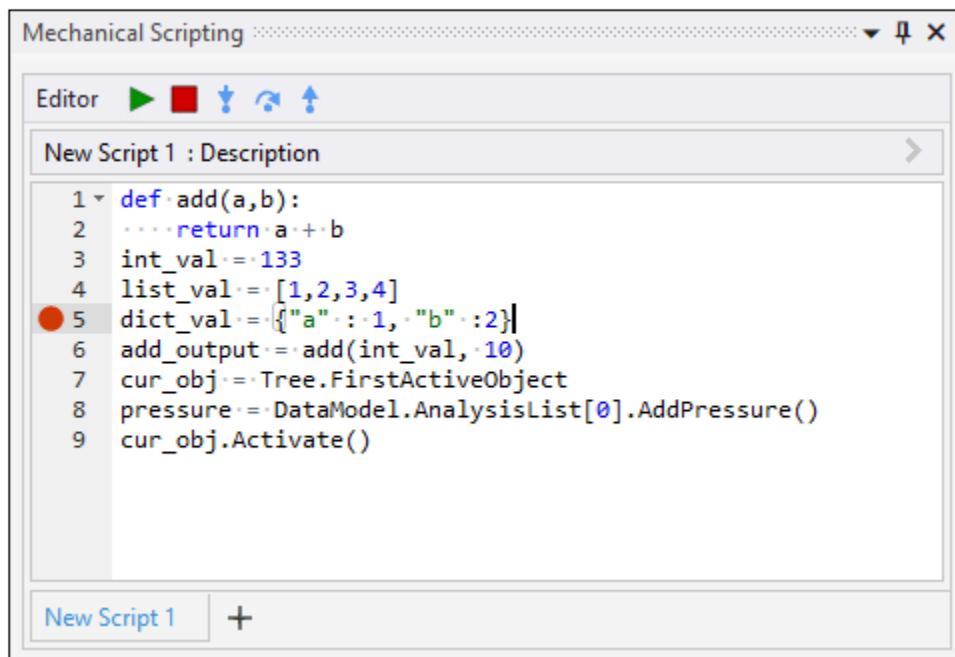
Debugging Scripts and Extensions

On the [Editor Toolbar \(p. 5\)](#), the drop-down menu of the **Run Script** button includes the **Start Debugger** action. This action starts executing script using the debugger in the **Mechanical Scripting** pane. Actions for this feature includes:

- [Inserting Breakpoints \(p. 24\)](#)
- [Using the Actions of the Debugging Toolbar \(p. 26\)](#)
- [Displaying Inspection Tooltips \(p. 27\)](#)
- [Examining Expressions using the Watch Window \(p. 27\)](#)
- [Viewing Debug Extensions \(p. 28\)](#)
- [Debugging Limitations \(p. 29\)](#)

Inserting Breakpoints

You can set breakpoints by selecting the field next to the line numbers (a red circle icon is displayed).



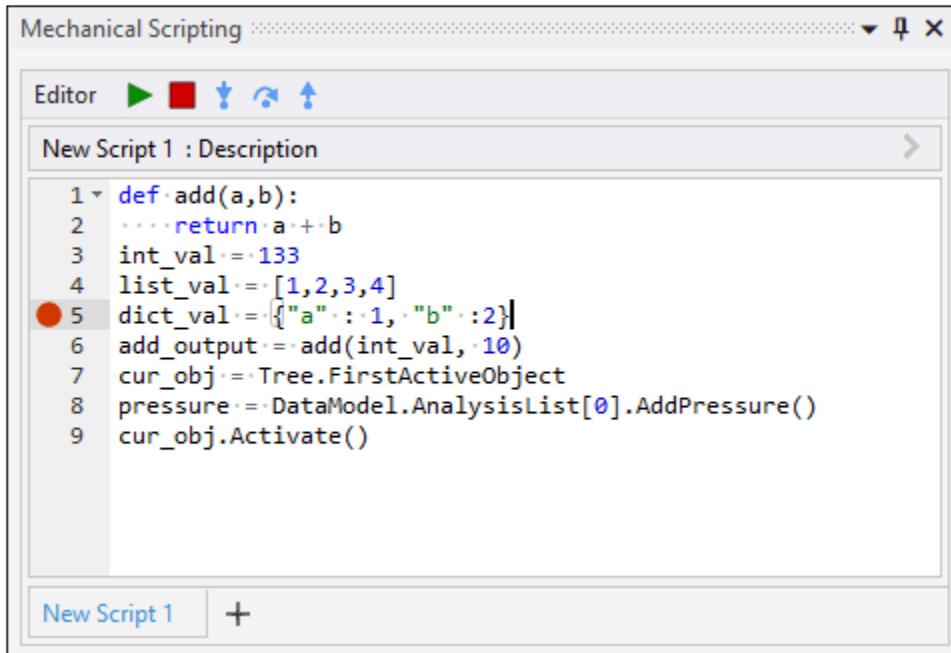
Important:

Breakpoints pause the script execution. When paused, you cannot interact with the Mechanical user interface.

Exceptions

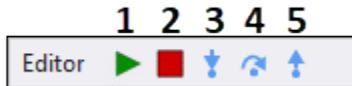
If there is an error in the script, an exception is thrown and a message displays describing the exception. Once you navigate from the message window, the line that includes the error is highlighted with a marker next to the line number. When you hover over this marker, you can read the description of the exception again.

When a script is run using the debugger and breakpoints have been inserted. The script will pause at the breakpoints as it executes. You can then use debugging toolbar actions to proceed. The current line of execution is highlighted in yellow, and an arrow is shown next to it.



Using the Actions of the Debugging Toolbar

Selecting the **Start Debugger** button displays the following debug toolbar.



Toolbar actions include:

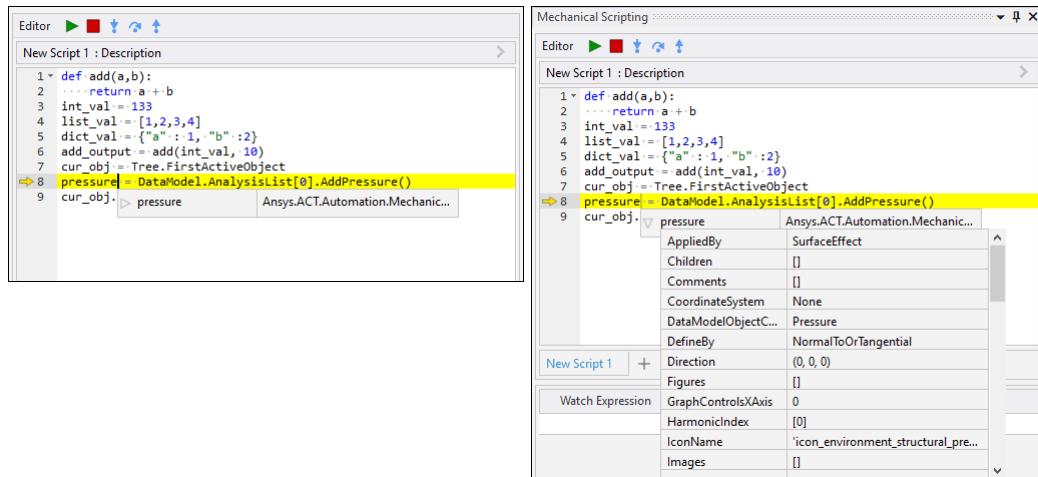
- Run Script/Continue (F5):** Continue execution to the next breakpoint when script is paused or re-start the debugging.
- Stop Debugging:** Exit debug mode.
- Step Into (F11):** Pauses at the first line of the function you are stepping into.
- Step Over (F10):** Execute the currently selected function or statement and then step over to the next line.
- Step Out (F12):** Continue execution to the next breakpoint or the next line after the function returns.

Note:

The availability of toolbar actions changes based on the current state of the debugger.

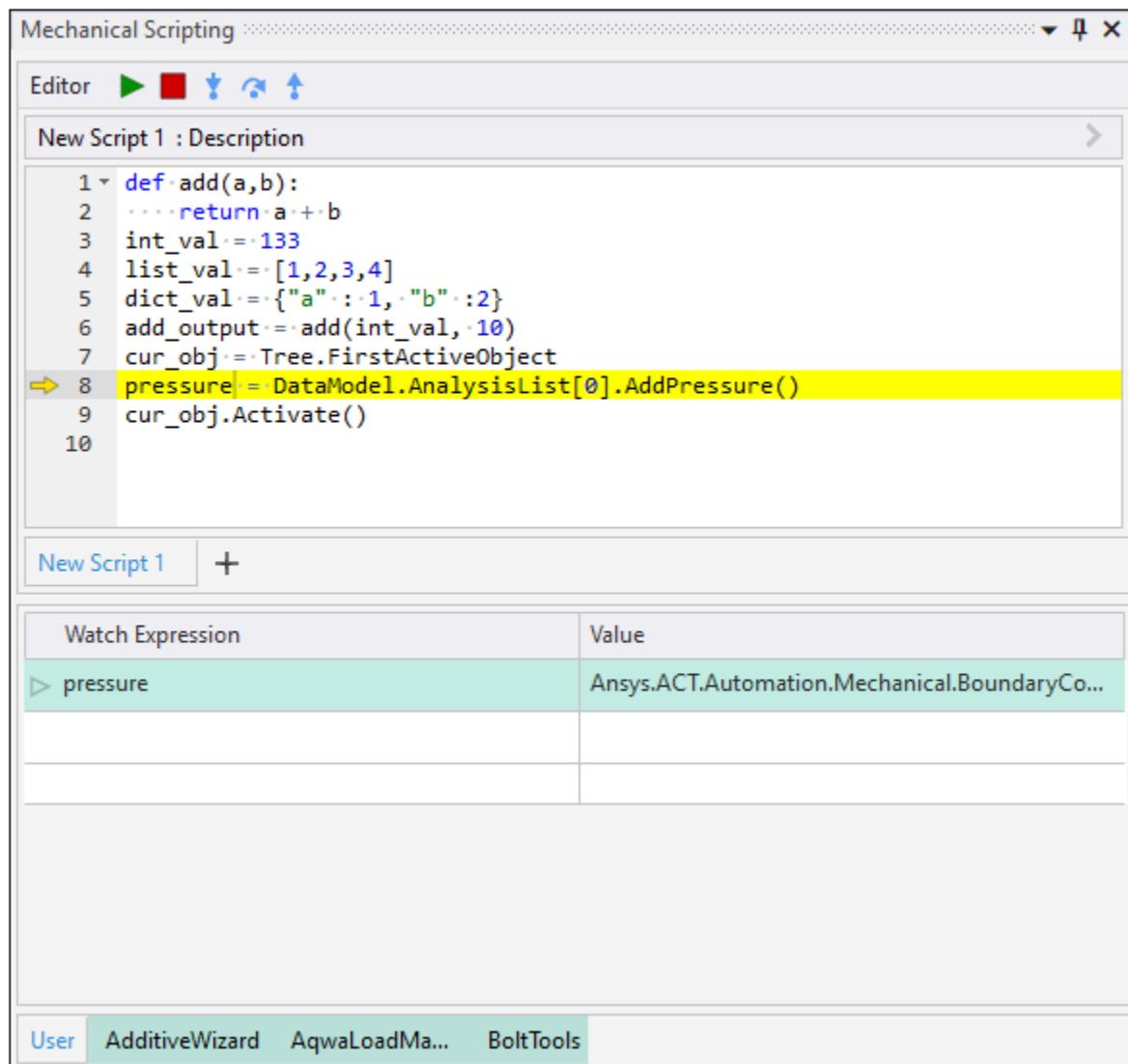
Displaying Inspection Tooltips

The inspection tooltip allows you to inspect variables when you are paused at a breakpoint. You can use the inspection tooltip by hovering over a variable. If a variable has additional properties, you can use the arrow beside the variable to display a drop-down table of the properties and values.



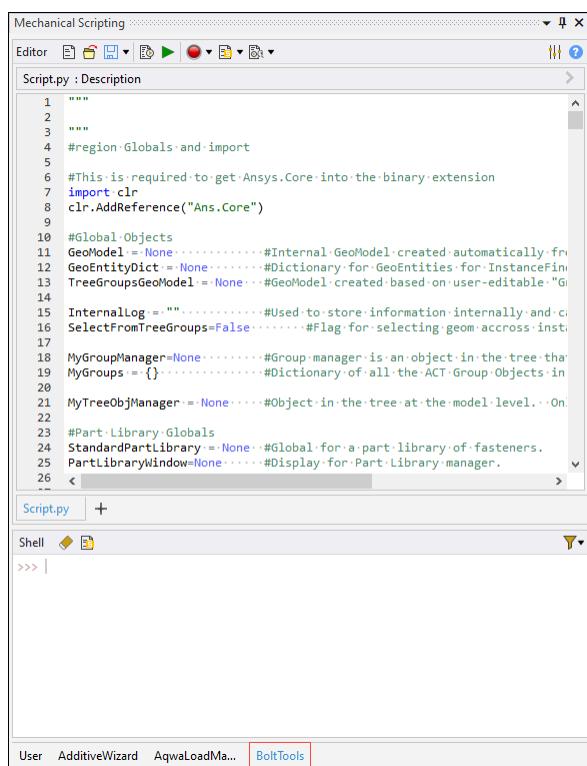
Examining Expressions using the Watch Window

When a script is run via the debugger, a watch window displays below the editor. To use this window, the editor must be paused at a breakpoint. The window enables you to enter an expression in the **Watch Expression** column evaluate expressions and "watch" their values change as you step through your script. Place the cursor in the **Watch Expression** column to enter and/or delete entries. Use the **[Tab]** key to execute your entries.



Viewing Debug Extensions

When you turn debug mode on in Workbench, you can use the same debug feature available in the **Mechanical Scripting** pane to [debug extension \(p. 29\)](#) python scripts. When you turn the feature on, you see the extension tabs at the very bottom of the **Shell** pane.



Debugging features are the same when working with an extension. Simply switch to the appropriate extension tab in the scripting view and you will see all python scripts in the editor. The only difference is, because extensions work based on callbacks, breakpoints are not encountered until you perform an action in the interface that executes the callback.

Important:

If you change the script of an extension, you must save and then reload the extension before rerunning/running the debugger. The application will prompt you to save the file if you make and change and then run the debug tool. However, if you make a change and select the Save button. You must also reload the extension using the **Reload** button in the **ACT Development** group on the **Automation** tab.

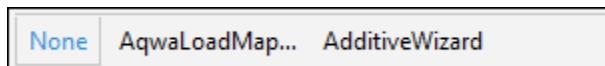
Debugging Limitations

If you run the command `import os` in the debugger, an exception is generated. To avoid this issue, you first need to turn on the **Debug Mode** option in the **Extensions** category of the Workbench **Options** dialog available through the **Tools** menu. Once selected, you need to execute the command using **Run Script** button. Following this you can use the debugging feature without producing an exception.

Scope Selection for ACT Extensions

Each ACT extension runs Python code in its own script scope. To access global variables or functions associated with an extension, you must first select the associated tab. There will be one tab for each extension loaded in Mechanical.

The following image shows the state of the tabs when supplied extensions **AqwaLoadMapping** and **AdditiveWizard** are loaded:



Scopes are only shown if debug mode is enabled for ACT extensions. For more information, see [Debug Mode in the ACT Developer's Guide](#).

Key Usage Concepts

Understanding these key concepts makes Mechanical's scripting easier to learn:

- Mechanical APIs can be used to access much of Mechanical, including tree objects and their properties.
- It is important to understand a key but subtle distinction between two APIs related to the mesh:
 - **Model.Mesh** accesses the object **Mesh** in the Mechanical tree, containing APIs for that object's properties in the **Details** view. It can be used to add and access mesh controls.
 - **DataModel.MeshByName ("Global")** accesses FE information (node and element locations, element connectivity, and so on).
- It is important to understand a key but subtle distinction between two APIs related to the geometry:
 - **DataModel.GeoData** accesses the underlying geometry attached to Mechanical.
 - **Model.Model.Geometry** accesses the object **Geometry** in the Mechanical tree.
- The underlying **Geometry**, much like the **Tree**, is hierarchical. For example, **DataModel.GeoData.Assemblies[0].Parts[0].Bodies[0].Volume** accesses the volume for the first part in the first (and only) assembly.
- All bodies have parts as their parents, even if they are not multibody parts. This is important to understand in both **GeoData** and when traversing the object **Geometry** of the **DataModel**. Although a part might be hidden from the Mechanical interface, the part is always there. For more information, see [Multibody Behavior and Associativity](#) in the *Ansys Mechanical User's Guide*.
- For more information on accessing the properties of an object, including those for traversing the geometry, mesh, simulation, and results, see [Mechanical APIs \(p. 37\)](#).
- It is often useful to store variables to access later rather than using the same API over and over. For example, using the following three commands is better than duplicating the **DataModel.MeshByName ("Global")** expression in the second and third commands:

```
mesh = DataModel.MeshByName("Global")
```

```
mesh.ElementCount
```

```
mesh.NodeCount
```

- Looping is a fundamental concept in any scripting language. Within the object **Geometry**, you can use loops to add, modify, or export property data. The following script loops over all contacts and change their formulation to MPC:

```
contact_region_list = DataModel.GetObjectsByType(DataModelObjectCategory.ContactRegion)
```

For **contact_region** in **contact_region_list**:

```
    contact_region.ContactFormulation = ContactFormulation.MPC
```

- The object **Transaction()** can be used to speed up a block of code that modifies multiple objects. This object ensures that the tree is not refreshed and that only the bare minimum graphics and validations occur while the transaction is in scope. The object **Transaction()** should only be used around code blocks that do not require state updates, such as solving or meshing. For example, to avoid redundant work from each addition, the following example use the object **Transaction()** before code that adds many objects:

```
with Transaction():
    for bodyId in bodyIds:
        ...
```

- If you cannot find what you want, check the attribute **InternalObject**. While the ACT API has “wrapped” many useful aspects of Mechanical, it has not wrapped everything, for various reasons. Many API objects have an **InternalObject** attribute that you can use to find additional capabilities that are not formally exposed in ACT.

For more information about how to complete many types of Mechanical tasks using scripts, see the examples in these sections:

- [Script Examples for Selection \(p. 163\)](#)
- [Script Examples for Interacting with Tree Objects \(p. 173\)](#)
- [Script Examples for Interacting with the Mechanical Session \(p. 193\)](#)

Threading

Although using Mechanical scripting APIs on a background thread might work in practice, the APIs are generally not thread-safe, and race conditions might and often do occur when trying to access them from a background thread.

It is still possible to only run parts of your code—namely those parts that do not use these APIs and hence do not risk race conditions—on a background thread. To do so, you can offload work to a new thread. A convenient way to do this is by using the `InvokeBackground` method exposed on `ExtAPI.Application`. This method can only take a function without any arguments, but the following technique can be used to pass in arguments to that function:

```
#function that is to be run in the background. It is safe because it does not use any of the Mechanical scripting APIs
def gradient(vectors)
    print("Computing gradients of the vectors")

#function that is run on the main thread
def my_script(result):
    vectors = some_function(result)
    def invokeInBackground():
        gradient(vectors)
    ExtAPI.Application.InvokeBackground(invokeInBackground)
```

Note:

A race condition is a software problem that can arise when concurrent code does not synchronize data access and mutation. These conditions are by their nature difficult to identify and reproduce, and they sometimes lead to seemingly random problems. There is no way to predict the outcome of a race condition. Alarmingly, it is possible for code to work well for years and then suddenly start to crash because of a race condition.

Additional Resources

To help you use scripts in Mechanical, many resources are available.

- The installed Python scripts that Mechanical uses can be insightful. These scripts are in your Ansys installation directory at `.../aisol/DesignSpace/DSPages/Python`. Useful scripts include `toolbar.py` and `selection.py`. These scripts are run when using many of the options in the **Select** group on the Mechanical ribbon's **Selection** tab. For more information, see [Selection Tab in the Ansys Mechanical User's Guide](#).
 - The [ACT API Reference Guide](#) provides descriptions of all ACT API objects, methods, and properties.
-

Note:

Ansys support is available anytime by emailing <support@ansys.com>.

Mechanical APIs

Mechanical APIs (Application Programming Interfaces) provide access to the native functionality of Ansys Mechanical.

[Mechanical API Introduction](#)

[Mechanical API Notes](#)

[Object Access](#)

[Boundary Conditions](#)

[Worksheets](#)

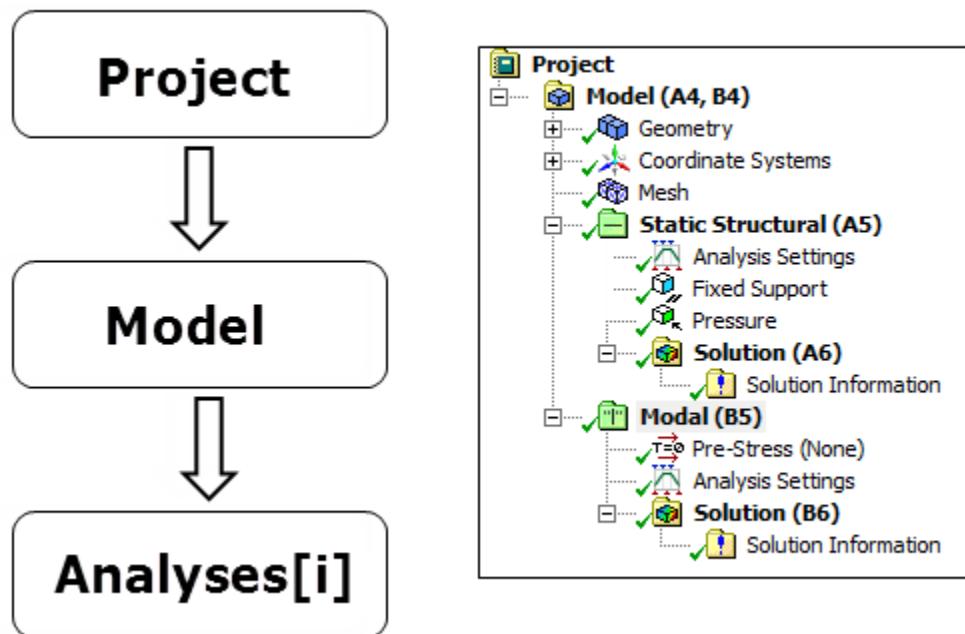
[Graphics](#)

[Results](#)

[Other APIs](#)

Mechanical API Introduction

Using APIs, you can access, modify, and add objects in the Mechanical tree (**Project**, **Model**, **Analyses**, and so on).



Note:

When you add an object using the API, the default values for properties in the **Details** view in Mechanical are the same as when you add an object directly in Mechanical.

Mechanical API Notes

The following topics provide Mechanical API migration notes and known issues and limitations:

[Mechanical API Migration Notes](#)

[Mechanical API Known Issues and Limitations](#)

Mechanical API Migration Notes

As improvements are made to Mechanical APIs and the way that they display and transmit data, great efforts are taken to ensure that changes are backwards-compatible. For your convenience, this section lists 2022 R2 Mechanical API changes that might impact your existing scripts so that you can determine if any action is necessary before migrating them.

Note:

For general ACT migration information, see [Migration Notes](#) in the *ACT Developer's Guide*.

Behavior change for Tree.AllObjects

For geometries in a Mechanical system that are linked to an **External Model** system, if a geometry in Mechanical has a single body part, the part name will be the same as the body name. Consequently, this query now returns two matches:

```
[x for x in DataModel.Tree.AllObjects if x.Name=="Surface Body 1(External Model)"]
```

To get the body, run either of these queries instead:

```
SURFACE_BODY1 = DataModel.Tree.Find(name="Surface Body 1(External Model))[0]
```

```
SURFACE_BODY1 = [x for x in DataModel.Tree.AllObjects if x.Name=="Surface Body 1(External Model)" and x.GetType()
```

Set/GetLoadCombinationType

- The behavior of SetLoadCombinationType/GetLoadCombinationType for the Solution Combination object has changed. The LoadCombinationType was previously an integer. Now the LoadCombinationType is set using an enum as shown below. In addition, GetLoadCombinationType returns an enum instead of integer

Original:

```
sc = Model.AddSolutionCombination()
scdef = sc.Definition
scdef.SetLoadCombinationType(0,1)
```

Migrated:

```
sc = Model.AddSolutionCombination()
scdef = sc.Definition
scdef.SetLoadCombinationType(0, LoadCombinationType.SRS)
```

Mechanical API Known Issues and Limitations

This section lists known issues and limitations related to Mechanical APIs.

Note:

For general ACT known issues and limitations, see [Known Issues and Limitations](#) in the *ACT Developer's Guide*.

General Issues and Limitations

ACT is unable to create a chart from Ansys Mechanical

When using ACT to create a figure from the chart API, the following error prevents the graphics display in the Mechanical window:

```
Object reference not set to an instance of an object.
```

As a workaround, add the following code to your script to create an empty window in which the chart can display:

```
import clr
clr.AddReference("Ans.UI.Toolkit")
clr.AddReference("Ans.UI.Toolkit.Base")
import Ansys.UI.Toolkit
if Ansys.UI.Toolkit.Window.MainWindow == None:
    Ansys.UI.Toolkit.Window.MainWindow = Ansys.UI.Toolkit.Window()
```

ExtAPI.Graphics.ExportScreenToImage() does not work if Mechanical is not open

If Mechanical is not open, `ExtAPI.Graphics.ExportScreenToImage()` does not work. This means that ACT cannot be used to export images from a DesignXplorer Design of Experiments or in batch mode.

Limitations on ACT Postprocessing of Mechanical Results

• Scoping for custom results defined in ACT extensions not supported

Custom results do not support using a geometric path as scoping. You can only use a selection of nodes and elements as scoping.

• Exporting external solver results to a text file

When exporting results obtained from an external solver to a text file, the **Include Node Location** option is not currently supported.

- **Solution Combination for external solvers**

You cannot currently combine results with the Solution Combination object to when using results from an external solver.

- **Compressed result file not supported**

The ACT postprocessing API does not support the compressed result file for Mechanical, which is created by the Mechanical APDL command **/FCOMP**.

- **A node merge action on the mesh is unsupported**

The ACT postprocessing API does not support a node merge action on the mesh.

- **ShellPosition command on shell bodies is unsupported**

The ACT postprocessing API does not account for the ShellPosition command for dealing with shell bodies.

- **Orientation nodes on beam bodies are not discarded**

The ACT postprocessing API returns the results on orientation nodes when dealing with beam bodies, even though these results are irrelevant.

- **Degenerated nodes are not discarded**

When an element degenerates, the ACT Postprocessing API does not discard the degenerated nodes, thereby resulting in some repeated result values. For example, when a HEX element (8 nodes) degenerates into a WEDGE element (6 nodes), the ACT Postprocessing API still returns result values on 8 nodes (out of which 2 values are repeated).

- **Substep changes require clean and re-solve**

If you change **Analysis Settings** substep entries, to see this change reflected in the ACT Custom results, you must run the **Clear Generated Data** option and re-solve your analysis.

Mechanical Limitations Unique to Running on Linux

- **Some property controls are not supported**

Mechanical does not support the following property controls on Linux:

- **FileOpen**
- **FolderOpen**
- **PropertyTable**

These controls are implemented using the Ansys UI Toolkit, which is currently not supported on Linux when executed within Mechanical.

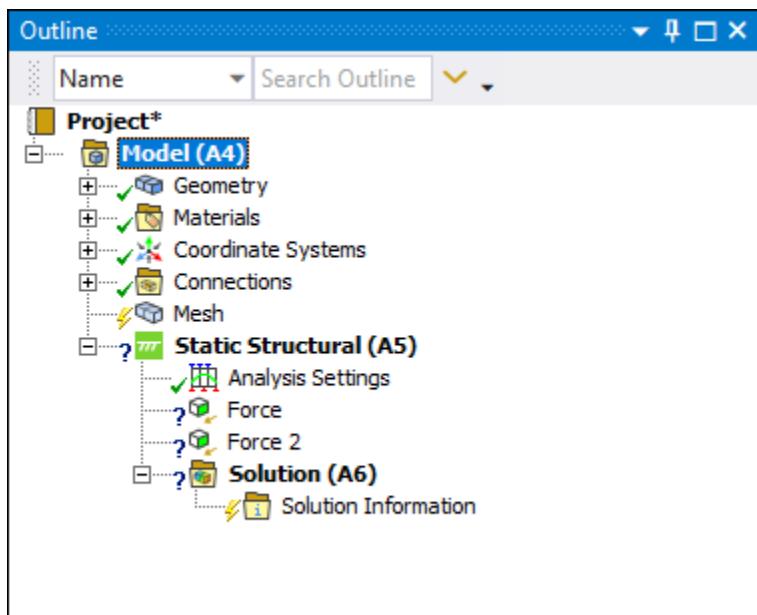
- **Graphics API issues in Mechanical when no extensions are loaded**

When no extensions are loaded, there are some limitations on the **Graphics** API from the **ACT Console** in Mechanical (and also in Ansys DesignModeler). For instance, Factory2D does not work. Therefore, you should load one or more extensions before using the **Graphics** API from the **ACT Console**.

Object Access

Using APIs, you can access Mechanical tree objects. The object representing the project node of the tree is **DataModel.Project**.

Each object can have children that you access by using the property **Children**. For example, assume that the **Outline** view in Mechanical looks like this:



To return all objects directly under the **Project** node, you enter this:

```
DataModel.Project.Children
```

Under the **Project** node is the **Model** node. Some examples follow for accessing **Model** child nodes:

```
Mesh = Model.Mesh
```

```
Connections = Model.Children[3]
```

To access all objects with a given name, you use the method **GetObjectsByName**:

```
DataModel.GetObjectsByName( "name" )
```

To access all objects if a given type, you use the method **GetObjectsByType** and pass the data model object category (such as **DataModelObjectCategory.Force**) as an argument:

```
DataModel.GetObjectsByType(DataModelObjectCategory.Force)
```

Property Types

Each object in the tree might have properties that display in the **Details** view. The APIs use a handful of types to refer to different kinds of properties. Descriptions and examples of the most common types follow.

Quantity: A real value with a unit typically related to a physical quantity.

```
my_object.ElementSize = Quantity("0.1 [m]")
```

Number (float and integer): A real or integer value.

```
my_object.TetraGrowthRate = 2
```

Boolean: Either **True** or **False**.

```
my_object.WriteICEMCFDFiles = True
```

Enumeration: A named value out of a set of possible named values.

```
mmesh_method.Algorithm = MeshMethodAlgorithm.PatchIndependent
```

Entity Selection (**ISelectionInfo** or **IMechanicalSelectionInfo**): A value representing a geometric or mesh selection. Some objects are valid selections (such as a Named Selection object) and can be used as values for these properties. To apply a list of entities directly, you can:

- Create an instance of **MechanicalSelectionInfo** and specify the IDs.
- Assign this instance to an entity selection property (such as **Location**).

```
my_selection = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
my_selection.Ids= [28,25]
My_object.Location = my_selection
```

Tree

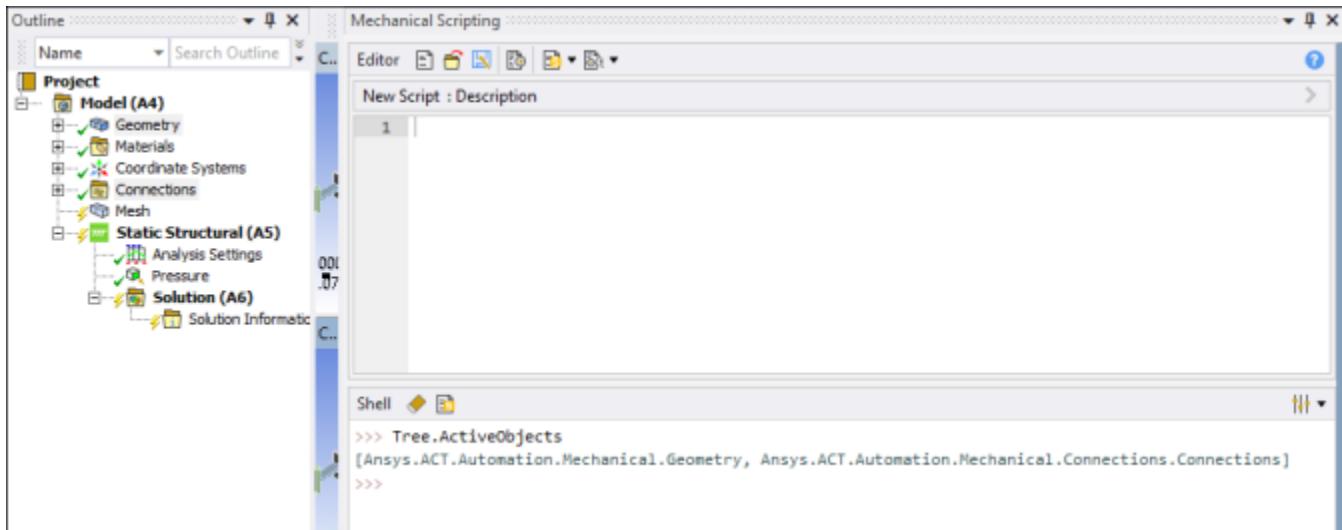
The **Tree** object provides access to the Mechanical tree.

To access this object, simply enter **Tree**. The following table provides a sampling of the APIs available on this object. Some usage examples appear after the table. For a comprehensive listing of methods and properties of the **Tree**, see [Tree](#) in the *ACT Online API and XML Reference Guide*.

Member	Description
ActiveObjects	Lists all selected objects. Read-only.
AllObjects	Lists all of the objects available in the tree. Read-only.
GetPathToFirstActiveObject	Shows the full statement that must be typed to get the selected object.
Refresh	Refreshes the tree.

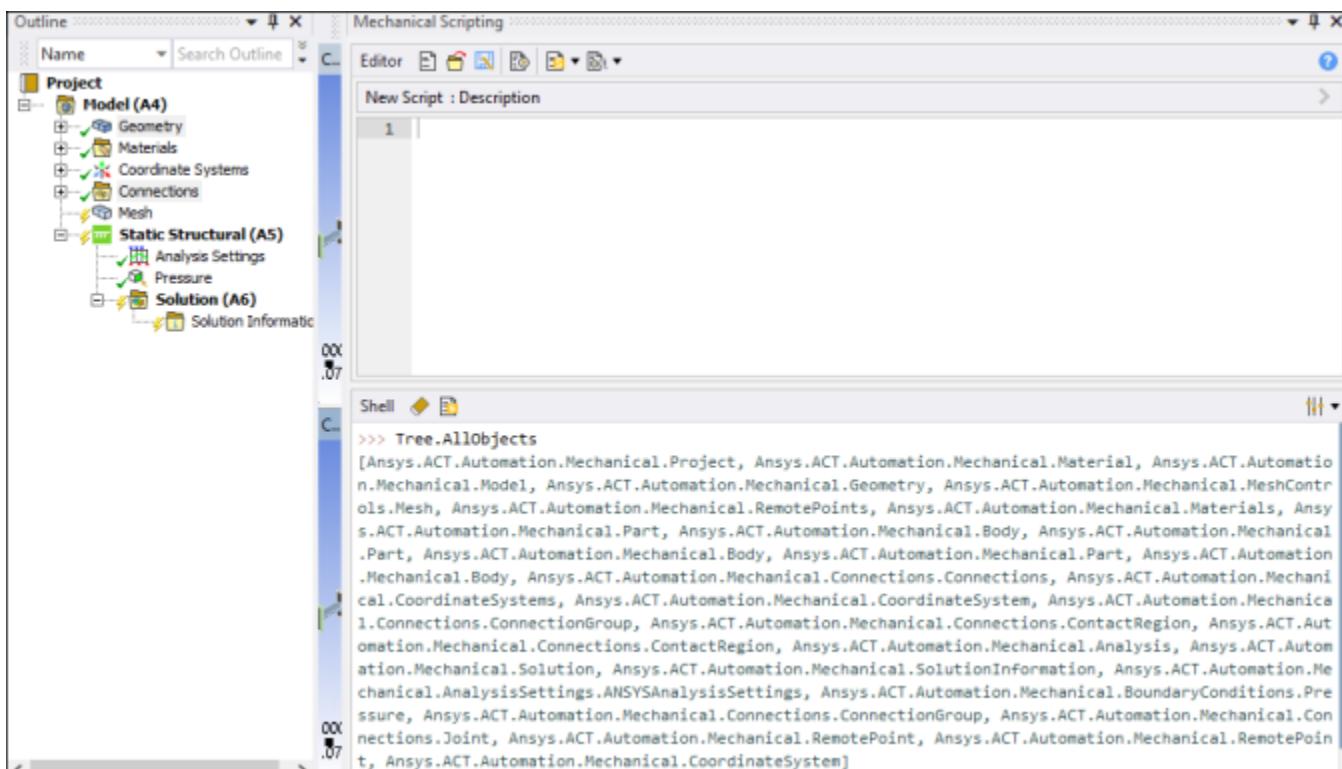
To access all objects selected in the tree:

```
Tree.ActiveObjects
```



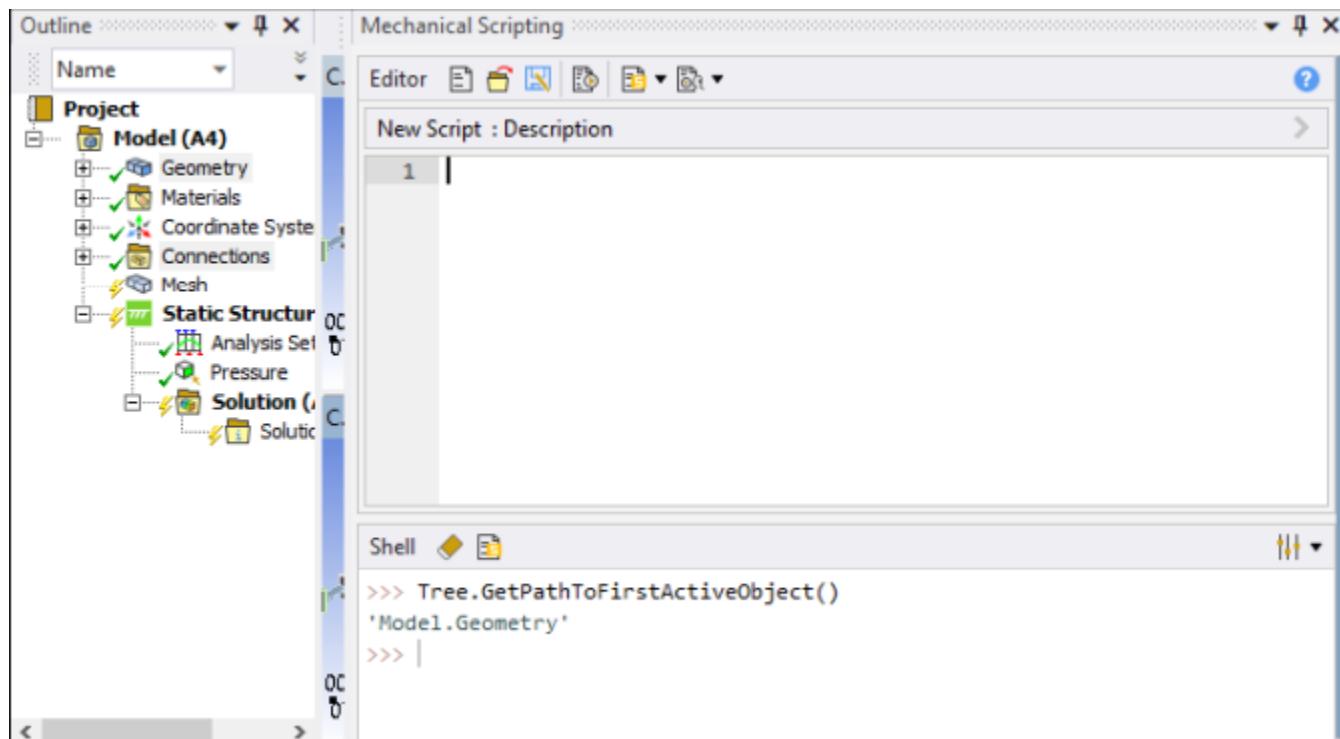
To access all objects in the tree:

```
Tree.AllObjects
```



To access the first object selected in the tree:

```
Tree.GetPathToFirstActiveObject()
```



Here are some other helpful APIs for performing tasks in the tree:

- Iterating

```
for obj in Tree
```

- Sort

```
Tree.Sort()
```

```
Tree.ClearSort()
```

- Filter

```
Tree.Filter(tag="tagname")
```

```
Tree.Filter(state=ObjectState.Suppressed)
```

```
Tree.Filter(visibility=False)
```

```
Tree.ClearFilter
```

- Find

```
objects = Tree.Find(name="substring", state=ObjectState.Unsuppressed)
```

```
for obj in objects:
```

- Events

```
def myFunc(sender, args):...
```

```
Tree.OnActiveObjectChanged += myFunc
```

- Grouping

```
Tree.Group([obj1, obj2, obj3, ...])  
  
Tree.HideAllGroupingFolders()  
  
Tree.ShowAllGroupingFolders()
```

- Multi-select

```
Tree.Activate([obj1, obj2, obj3, ...])
```

Model Objects

The following topics describe the objects **Geometry**, **Mesh**, **Connections**, and **Analysis** and how you can access and manipulate them:

- [Accessing and Manipulating the Geometry Object](#)
- [Accessing and Manipulating the Mesh Object](#)
- [Accessing and Manipulating the Connections Object](#)
- [Accessing and Manipulating the Analysis Object](#)

Accessing and Manipulating the Geometry Object

The **Geometry** object provides an API for the **Geometry** tree object.

To access the **Geometry** object:

```
geometry = Model.Geometry
```

The **Geometry** object exposes several convenient methods for adding child objects. For example, you can add a point mass to the **Geometry** object by calling the method **AddPointMass**.

```
point_mass = geometry.AddPointMass()
```

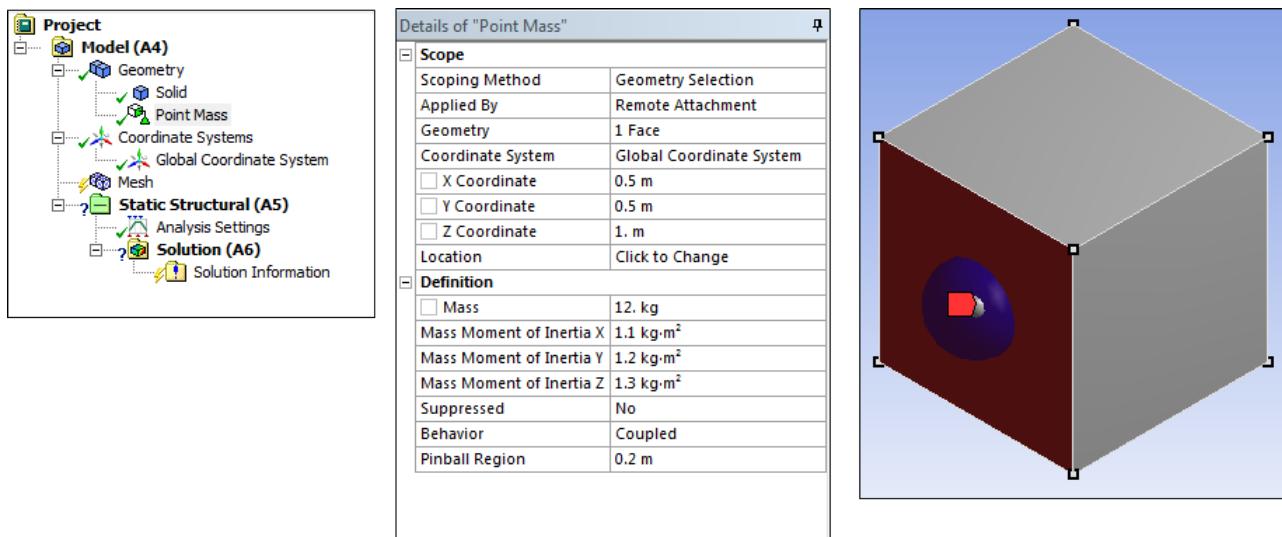
For this point mass, you can assign a location:

```
my_selection = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
my_selection.Ids = [22]
point_mass.Location = my_selection
```

For the above point mass, accessible properties include:

```
point_mass.Mass = Quantity("12 [kg]")
point_mass.MassMomentOfInertiaX = Quantity("1.1 [kg m m]")
point_mass.MassMomentOfInertiaY = Quantity("1.2 [kg m m]")
point_mass.MassMomentOfInertiaZ = Quantity("1.3 [kg m m]")
point_mass.Behavior = LoadBehavior.Coupled
point_mass.PinballRegion = Quantity("0.2 [m]")
```

Combining the three previous actions, the geometry now contains a fully defined point mass.



You can export the **Geometry** object to an STL (STereoLithography) file, which is the most commonly used file format in 3D printing. The following command exports the geometry object to an STL file.

```
Model.Geometry.ExportToSTL("C:\Temp\geoasst1.stl")
```

The result is the creation of a geometry file (geoasst1.stl) to the fully qualified directory path (C:\Temp).

Accessing and Manipulating the Mesh Object

The **Mesh** object provides an API for the **Mesh** tree object.

To access the **Mesh** object:

```
mesh = Model.Mesh
```

The **Mesh** object exposes several convenient methods to add mesh controls. For example, you can create a meshing control that applies a patch-independent algorithm to the mesh by calling the method **AddAutomaticMethod**.

```
mesh_method = mesh.AddAutomaticMethod()
```

Mesh control objects often require a valid scoping. You can satisfy this requirement by setting the **Location** property:

```
my_selection = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
my_selection.Ids = [16]
mesh_method.Location = my_selection
```

For the above mesh control, accessible properties include:

```
mesh_method.Method = MethodType.AllTriAllTet
mesh_method.Algorithm = MeshMethodAlgorithm.PatchIndependent
mesh_method.MaximumElementSize = Quantity("0.05 [m]")
mesh_method.FeatureAngle = Quantity("12.00000000000002 [degree]")
```

```

mesh_method.MeshBasedDefeaturing = True
mesh_method.DefeaturingTolerance = Quantity("0.0001 [m]")
mesh_method.MinimumSizeLimit = Quantity("0.001 [m]")
mesh_method.NumberOfCellsAcrossGap = 1
mesh_method.CurvatureNormalAngle = Quantity("36 [degree]")
mesh_method.SmoothTransition = True
mesh_method.TetraGrowthRate = 1

```

Weld allows scripting. Weld scripting allows you to view and edit the erroring welds easily when working with complex models. The **Weld** control functions can be accessed as follows:

To access **Weld** control, you should enable **Batch Connections**.

To enable **Batch Connections**:

```
mesh.MeshBasedConnection = True #Enables the Batch Connections.
```

To insert **Weld** control:

```
weld = mesh.AddWeld() #Adds the weld control.
```

To access the **Weld** options:

```

weld.ControlType = WeldType.Seam #Sets the Weld Type as Continuous Seam (Seam). The other option available for
weld.Source = WeldSource.Geometry #Sets the Weld Source as Geometry. The other option available for Weld Source
weld.ModeledAs = WeldModeledAs.TentAndExtension #Sets the WeldModeledAs as Normal and Angled (Tent and Extension)
weld.CreateUsing = WeldCreateUsing.Curves #Sets the WeldCreateUsing as Curves. This option is only available when
Weld.TentDirection = WeldTentDirection.Normal #Sets the direction of tent. The other option available for WeldTentDirection
weld.UseWorksheet = YesNoType.No #Enables the Worksheet if set to Yes.
weld.Suppressed = False
weld.AdjustWeldHeight = YesNoType.No #Enables AdjustWeldHeight if set to Yes.
weld.CreationCriteria = WeldCreationCriteria.Width #Sets the Creation Criteria as Width. The other option available
weld = DataModel.GetObjectById(object_id) #Gets the object by its id.
weld.EdgeMeshSize = Quantity(9.0, "mm")#Provides the EdgeMeshSize.
weld.CreateOffsetLayer = YesNoType.Yes #Creates offset layers if set to Yes.
weld.OffsetLayerHeight = Quantity(9.0, "mm") #Sets the OffsetLayerHeight.
weld.NumberOfLayers = 1 #Sets the number of layers.
weld.OffsetLayerGrowthRate = 1.2 #Set the growth rate for offset layer.
weld.GenerateEndCaps = YesNoType.Yes #Generates triangular end-caps if set to Yes at the free ends of the welds.
weld.GenerateNamedSelection = WeldGeneratedNamedSelection.No # Sets the Generated NamedSelection. The other option available
weld.MaterialId = None # Sets the material by its id. Here, 0 denotes None and 10 denotes Structural Steel.
weld.ThicknessAssignment = WeldThickness.Automatic #Sets the weld thickness. The other options available for weld.ThicknessAssignment
weld.SharpAngle = Quantity(30, "deg") #Set the Sharp Angle. Here, the default angle is 30 degrees. You can set the angle to any value between 0 and 180 degrees.
weld.ConnectionTolerance = Quantity(0, "mm")#Sets the Connection Tolerance. The default value is 0. You can set the tolerance to any value between 0 and 1000 mm.
weld.Smoothing = YesNoType.Yes #Set the Smoothing to Yes by default.

```

Accessing and Manipulating the Connections Object

The **Connections** object provides an API for the **Connections** tree object.

To access the **Connections** object:

```
connections = Model.Connections
```

The **Connections** object exposes several convenient methods for adding connections. For example, you can add a beam or set the contact type to frictionless on one of the model's contact regions:

```
beam = connections.AddBeam()
```

```
contact_region = connections.Children[0].Children[0]
contact_region.ContactType = ContactType.Frictionless
```

Beam objects require a valid scoping. You can satisfy this requirement by setting the appropriate property. For a beam, you set the **ReferenceLocation** and **MobileLocation** properties:

```
reference_scoping = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
reference_scoping.Ids = [110]
beam.ReferenceLocation = reference_scoping
mobile_scoping = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
mobile_scoping.Ids = [38]
beam.MobileLocation = mobile_scoping
```

For the above beam, accessible properties include:

```
beam.ReferenceBehavior = LoadBehavior.Deformable
beam.ReferencePinballRegion = Quantity("0.001 [m]")
beam.Radius = Quantity("0.005 [m]")
beam.MobileZCoordinate = Quantity("6.5E-03 [m]")
beam.MobilePinballRegion = Quantity("0.001 [m]")
```

Accessing and Manipulating the Analysis Object

The **Analysis** object provides an API for the **Environment** tree object.

To access the first **Analysis** object in the tree and its settings:

```
analysis1 = Model.Analyses[0]
analysis_settings = analysis1.AnalysisSettings
```

The **Analysis** object exposes several convenient methods. For example, you can add boundary conditions like bolt pretensions, loads, and fixed supports:

```
bolt = analysis1.AddBoltPretension()
pressure = analysis1.AddPressure()
force = analysis1.AddForce()
support = analysis1.AddFixedSupport()
```

Boundary condition objects often require a valid scoping. You can satisfy this requirement by setting **Location** properties:

```
pressure_scoping = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
pressure_scoping.Ids = [220]
pressure.Location = pressure_scoping
force_scoping = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
force_scoping.Ids = [219]
force.Location = force_scoping
```

For the above boundary conditions, accessible properties include:

```
bolt.SetDefineBy(1, BoltLoadDefineBy.Load) # Change definition for step #1.
bolt.Preload.Output.SetDiscreteValue(0, Quantity("15 [N]")) # Change preload value for step #1.
pressure.Magnitude.Output.Formula = '10*time' # To use a formula
pressure.Magnitude.Output.DiscreteValues=[Quantity('6 [Pa]')] # To use a direct value
force.Magnitude.Output.DiscreteValues=[Quantity('11.3 [N]'), Quantity('12.85 [N]')]
```

Analysis Settings Object

For the **Analysis Settings** object, you can Get or Set the step-dependent properties of certain object categories without changing the current step number. This includes the **Step Controls**, **Nonlinear Controls**, and the **Output Controls** categories, as described below, of the Details properties of the object.

Step Controls

For the **Step Controls**, you can modify all of the properties without the need to change the **Current Step Number** property.

Nonlinear Controls

For the **Nonlinear Controls** category, you can modify the various convergence properties that are "step aware" (can vary per step) as well as Stabilization.

Output Controls

For the **Output Controls** category, you can modify the **Store Results At** property and its accompanying **Value** property.

Note:

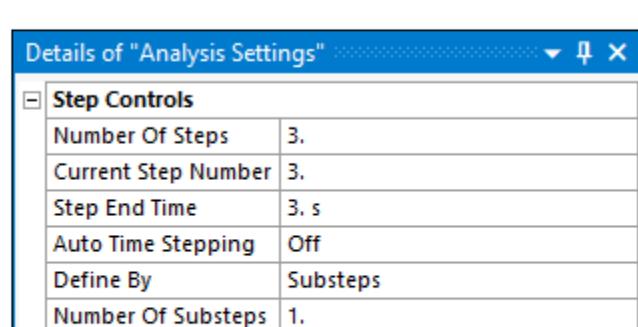
Review the [Analysis Settings](#) section for the above categories as well as the [Steps and Step Controls for Static and Transient Analyses](#) section of the *Mechanical User's Guide* for additional information about step-based properties and analyses.

Script Access and Examples

The script to access the available properties of the Analysis Settings object:

```
analysis_settings = Model.Analyses[0].AnalysisSettings
```

This example modifies the **Auto Time Stepping** property.



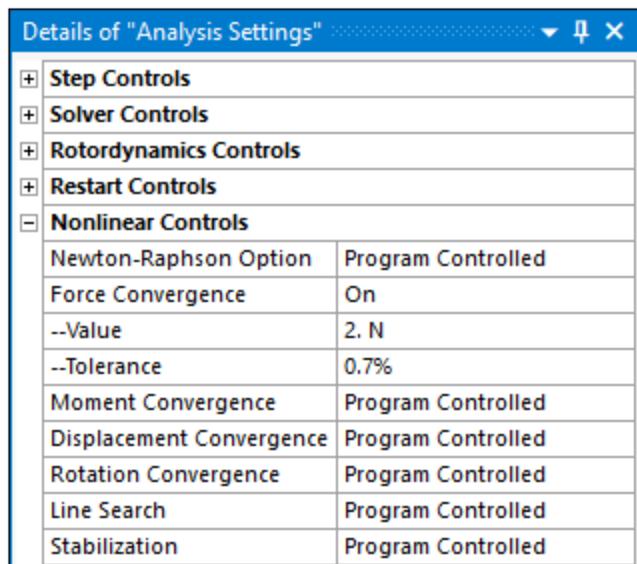
Shell

```

>>> analysis_settings = Model.Analyses[0].AnalysisSettings
>>> analysis_settings.SetAutomaticTimeStepping(1, Automatic)
>>> analysis_settings.GetAutomaticTimeStepping(1)
On
>>> analysis_settings.GetAutomaticTimeStepping(2)
ProgramControlled
>>> analysis_settings.SetAutomaticTimeStepping(3, Automatic)
>>> analysis_settings.GetAutomaticTimeStepping(3)
Off
>>>

```

This example modifies the **Force Convergence** property and sets its corresponding **Value** and **Tolerance** properties.



```
Shell ◻ E
>>> analysis_settings = Model.Analyses[0].AnalysisSettings
>>> analysis_settings.SetForceConvergenceType(2, ConvergenceType.Force)
>>> analysis_settings.GetForceConvergenceType(2)
On
>>> analysis_settings.SetForceConvergenceValue(2, Quantity(2, 'N'))
>>> analysis_settings.GetForceConvergenceValue(2)
'2 [N]'
>>> analysis_settings.SetForceConvergenceTolerance(2, 0.7)
>>> analysis_settings.GetForceConvergenceTolerance(2)
0.7
>>> |
```

This example modifies the **Store Results At** property and sets its corresponding **Value** property.



```
Shell ◻ E
>>> analysis_settings = Model.Analyses[0].AnalysisSettings
>>> analysis_settings.SetStoreResultsAt(2, TimePointsOptions.EquallySpacedPoints)
>>> analysis_settings.GetStoreResultsAt(2)
1
>>> |
```

Object Traversal

The following topics describe how to traverse the objects **Geometry**, **Mesh**, and **Results**:

[Traversing the Geometry](#)

Traversing the Mesh

Traversing Results

Note:

- The sample code in this section is taken from the supplied extension TraverseExtension. You can download the package of extension examples from the developer help panel for the [ACT Start Page](#).
- For comprehensive information on interfaces and properties, see the [Ansys ACT API Reference Guide](#).

Traversing the Geometry

The APIs for geometry data provide access to the underlying geometry that is attached in Mechanical. For example, the API could be used to determine the faces associated with an edge.

The basic hierarchy of the geometry is:

```
- Geometry
  - Assembly
    - Part
      - Body
        - Shell
        - Face
        - Edge
        - Vertex
```

An example Python function to traverse geometry data follows. Here, an object of type **IGeoData** is obtained from the object **Analysis** using the property **GeoData**. The object **GeoData** is then used to access the list of **IGeoAssembly** with the property **Assembly**. For each of the objects in this list, the property **Parts** is used to access the list of **IGeoPart** objects on the assembly. This pattern is repeated through the hierarchy of the geometry down to the vertices of each edge.

```
def traversegeometry(analysis):
    now = datetime.datetime.now()
    f = open("C:\\\\geoDump.txt", 'w')
    f.write("*.*.*.*.*.*\n")
    f.write(str(now)+"\n")
    # --- IGeometry Interface
    # +++ Properties and Methods
    # +++ Assemblies
    # +++ CellFromRefId
    # +++ SelectedRefIds
    geometry = analysis.GeoData
    assemblies = geometry.Assemblies
    assemblies_count = assemblies.Count
    # --- IGeoAssembly Interface
    # +++ Properties and Methods
    # +++ Name
    # +++ Parts
    for assembly in assemblies:
        assembly_name = assembly.Name
        parts = assembly.Parts
        parts_count = parts.Count
        # --- IGeoPart Interface
        # +++ Properties and Methods
        # +++ Name
```

```

# +++ Bodies
for part in parts:
    part_name = part.Name
    bodies = part.Bodies
    bodies_count = bodies.Count
    # --- IGeoBody Interface
    # +++ Properties and Methods
    # +++ Name
    # +++ Vertices
    # +++ Edges
    # +++ Faces
    # +++ Shells
    # +++ Material
    for body in bodies:
        faces = body.Faces
        faces_count = faces.Count
        # --- IGeoFace Interface
        # +++ Properties and Methods
        # +++ Body
        # +++ Shell
        # +++ Vertices
        # +++ Edges
        # +++ Loops
        # +++ Area
        # +++ SurfaceType
        # +++ PointAtParam
        # +++ PointsAtParams
        for face in faces:
            edges = face.Edges
            edges_count = edges.Count
            # --- IGeoEdge Interface
            # +++ Properties and Methods
            # +++ Faces
            # +++ Vertices
            # +++ StartVertex
            # +++ EndVertex
            # +++ Length
            # +++ CurveType
            # +++ Extents
            # +++ IsParamReversed
            # +++ ParamAtPoint
            # +++ PointAtParam
            # +++ PointsAtParams
            for edge in edges:
                vertices = edge.Vertices
                vertices_count = vertices.Count
                # --- IGeoVertex Interface
                # +++ Properties and Methods
                # +++ Edges
                # +++ Faces
                # +++ Bodies
                # +++ X
                # +++ Y
                # +++ Z
                for vertex in vertices:
                    xcoord = vertex.X
                    ycoord = vertex.Y
                    zcoord = vertex.Z
                    try:
                        f.write("      Vertex: "+vertex.ToString()+" , X = "+xcoord.ToString()+" , Y = "+ycoord.ToString())
                    except:
                        continue
            f.close()
        return
    
```

Traversing the Mesh

The API for mesh data provides access to the underlying mesh in Mechanical.

An example Python function to traverse mesh data follows. Here, an object of type **IMeshData** is obtained from the object **Analysis** using the property **MeshData**. The object **IMeshData** is then used to access the list of element IDs with the property **Elements**. Using each of the element IDs in the returned list, the method **ElementById** is called to access the element data with **IElement**. This pattern is repeated for all of the nodes for each element. Finally, the coordinates of the nodes are queried.

```
import datetime
def traversemesh(analysis):
    now = datetime.datetime.now()
    f = open("C:\\\\MeshDump.txt", 'w')
    f.write(".*.*.*.*.*.*\n")
    f.write(str(now)+"\n")
    # --- IMesh Interface
    # +++ Properties and Methods
    # +++ MeshRegion
    # +++ Node
    # +++ Element
    # +++ Nodes
    # +++ Elements
    # +++ NumNodes
    # +++ NumElements
    mesh = analysis.MeshData
    elementids = mesh.ElementIds
    # --- IElement Interface
    # +++ Properties and Methods
    # +++ Id
    # +++ Type
    # +++ Nodes
    for elementid in elementids:
        element = mesh.ElementById(elementid)
        nodeids = element.NodeIds
        # --- INode Interface
        # +++ Properties and Methods
        # +++ Id
        # +++ X
        # +++ Y
        # +++ Z
        # +++ Elements
        for nodeid in nodeids:
            node = mesh.NodeById(nodeid)
            nodex = node.X
            nodey = node.Y
            nodez = node.Z
            try:
                f.write("      Element: "+elementid.ToString()+"  Node: "+nodeid.ToString()+", X = "+nodex.ToString())
            except:
                continue
    f.close()
    return
```

Another example of traversing the mesh data follows. Only the elements of user-selected geometry entities are considered. The property **CurrentSelection** on the Selection Manager can be used to query the IDs of the selected geometry entities.

```
def elementcounter(analysis):
    with open("C:\\\\SelectedMeshEntities.txt", 'w') as f:
        geometry = analysis.GeoData
        smgr = ExtAPI.SelectionManager
        selectedids = smgr.CurrentSelection.Ids
        mesh = analysis.MeshData
        if selectedids.Count == 0:
            f.write("Nothing Selected!")
            return
        for selectedid in selectedids:
            entity = geometry.GeoEntityById(selectedid)
```

```
meshregion = mesh.MeshRegionById(selectedid)
try:
    numelem = meshregion.ElementCount
    f.write("Entity of type: "+entity.Type.ToString()+
           " contains "+numelem.ToString()+
           " elements.")
except:
    f.write("The mesh is empty!")
return
```

Traversing Results

The API for direct result access allows you to do postprocessing without result objects.

An example python function to compute minimum and maximum results follows. It begins by instantiating a result reader using the method `analysis.GetResultsData()`. Results are retrieved relative to the finite element model and queried using either the `elementID` (elemental result) or the `nodeID` (nodal result). The displacement result `U` is a nodal result, whereas the stress result `S` is a result on nodes of the elements. The displacement result stores a set of component values for each node, where the component names are X, Y, and Z.

The function first iterates over the `nodeIDs` to compute the minimum and maximum values. It then iterates over the `elementIDs` and the nodes of each element to compute the minimum and maximum values.

Note:

The second loop over the nodes is filtered to the corner nodes of the elements because stress results are available only on these corner nodes.

Finally, the results are written to the output file.

```
def minmaxresults(analysis):
    now = datetime.datetime.now()
    f = open("C:\\\\resultsInfo.txt", 'w')
    f.write("*.*.*.*.*.*\n")
    f.write(str(now)+"\n")
    #
    # Get the element ids
    #
    meshObj = analysis.MeshData
    elementids = meshObj.ElementIds
    nodeids = meshObj.NodeIds
    #
    # Get the results reader
    #
    reader = analysis.GetResultsData()
    reader.CurrentResultSet = int(1)
    #
    # Get the displacement result object
    displacement = reader.GetResult("U")

    num = 0
    for nodeid in nodeids:
        #
        # Get the component displacements (X Y Z) for this node
        #
        dispvals = displacement.GetNodeValues(nodeid)
        #
        # Determine if the component displacement (X Y Z) is min or max
        #
```

```

if num == 0:
    maxdispX = dispvals[0]
    mindispX = dispvals[0]
    maxdispY = dispvals[1]
    mindispY = dispvals[1]
    maxdispZ = dispvals[2]
    mindispZ = dispvals[2]

num += 1

if dispvals[0] > maxdispX:
    maxdispX = dispvals[0]
if dispvals[1] > maxdispY:
    maxdispY = dispvals[1]
if dispvals[2] > maxdispZ:
    maxdispZ = dispvals[2]
if dispvals[0] < mindispX:
    mindispX = dispvals[0]
if dispvals[1] < mindispY:
    mindispY = dispvals[1]
if dispvals[2] < mindispZ:
    mindispZ = dispvals[2]

# Get the stress result object
stress = reader.GetResult("S")

num = 0
for elementid in elementids:
    element = meshObj.ElementById(elementid)
    #
    # Get the SXX stress component
    #
    stressval = stress.GetElementValues(elementid)
    #
    # Get the primary node ids for this element
    #
    nodeids = element.CornerNodeIds
    for i in range(nodeids.Count):
        #
        # Get the SXX stress component at node "nodeid"
        #
        SXX = stressval[i]
        #
        # Determine if the SXX stress component is min or max
        #
        if num == 0:
            maxsxx = SXX
            minsxx = SXX

        if SXX > maxsxx:
            maxsxx = SXX
        if SXX < minsxx:
            minsxx = SXX

    num += 1
#
# Write the results to the output
#
f.write("Max U,X:Y:Z = "+maxdispX.ToString()+" : "+maxdispY.ToString()+" : "+maxdispZ.ToString()+"\n")
f.write("Min U,X:Y:Z = "+mindispX.ToString()+" : "+mindispY.ToString()+" : "+mindispZ.ToString()+"\n")
f.write("Max SXX = "+maxsxx.ToString()+"\n")
f.write("Min SXX = "+minsxx.ToString()+"\n")
f.close()

```

Property APIs for Native Tree Objects

A native tree object is not defined by an ACT extension. You can use these APIs as another way to interact with properties shown in the **Details** view. For example, assume that you have stored a variable named **rectbar** of a **Body** object with these properties shown in the **Details** view:

Details of "rectbar"	
+ Graphics Properties	
- Definition	
<input type="checkbox"/> Suppressed	No
Stiffness Behavior	Flexible
Coordinate System	Default Coordinate System
Reference Temperature	By Environment
Behavior	None
- Material	
Assignment	Structural Steel
Nonlinear Effects	Yes
Thermal Strain Effects	Yes
+ Bounding Box	
+ Properties	
+ Statistics	

To see the properties visible in the **Details** view, you enter:

```
rectbar.VisibleProperties
```

To list all properties that could be in the **Details** view for an object of this type, including those that are hidden, you enter:

```
rectbar.Properties
```

You can use property APIs to get the property's unique internal name, caption, internal value, and whether it is valid, visible, or read-only. You can also use **APIName** from the property APIs to see which API can be used directly on **rectbar** for the given **Details** view property, if an API exists.

Tip:

To change the internal value for any property, even one not yet wrapped by an API, you can specify the property's index value and then the current value and replacement value, entering something like this:

```
obj.Properties[0].InternalValue = newValue
```

For a scripting example, see [Get All Visible Properties for a Tree Object \(p. 174\)](#).

Details View Parameters

You can write scripts that create, query, or remove the parametrization of a property in the **Details** view of a native tree object. For example, in Mechanical, to the right of any property that can be parametrized, you can click the box, which causes a **P** to display, indicating that it is now a parameter.

Details of "rectbar"	
Graphics Properties	
Visible	Yes
Transparency	1
Color	
Definition	
<input checked="" type="checkbox"/> Suppressed	No
Stiffness Behavior	Flexible
Coordinate System	Default Coordinate System
Reference Temperature	By Environment
Behavior	None

This also adds the **Parameter Set** bar to the Workbench **Project Schematic**. Double-clicking the **Parameter Set** bar opens it so that you can see all input and output parameters in the project.

To parametrize a native property using an API, you use the method:

```
CreateParameter( "PropertyName" )
```

This method returns an instance of the *DetailsViewParameter* class. It can be used to query the Workbench ID for the parameter in the **Parameter Set** bar. You can also query the object and property name for which the parameter was created:

```
ID GetParameter( name )
Object GetParameter( name )
PropertyName GetParameter( name )
```

To no longer have a property be a parameter, you use the method:

```
RemoveParameter( "PropertyName" )
```

For a scripting example, see [Parametrize a Property for a Tree Object \(p. 174\)](#).

Solver Data

You can use **SolverData APIs** to get global items applicable to the solver or solver-related data specific to an object. Currently, **SolverData** APIs are available only for the MAPDL solver.

You use the **Solution** object to get solver data:

```
solution = Model.Analyses[0].Solution
solver_data = solution.SolverData
```

The following topics describe ways of using **SolverData** APIs to get global items and object-specific data:

- [Getting Global Items for Solver Data](#)
- [Getting Object-Specific Data](#)
- [Getting Object-Specific Data for Imported Objects](#)

Getting Global Items for Solver Data

You can use **SolverData** APIs to get the maximum element ID, maximum node ID, and maximum element type for the Ansys solver target.

- To get the maximum element ID:

```
solver_data.MaxValueId
```

- To get the maximum node ID:

```
solver_data.MaxValueId
```

- To get the maximum element type:

```
solver_data.MaxValueType
```

- To get the node IDs associated with a material ID:

```
solver_data.NodeIdsByMaterialId(int matId)
```

- To get the element IDs associated with a material ID:

```
solver_data.ElementIdsByMaterialId(int matId)
```

Important:

The following functions should be used with the background thread.

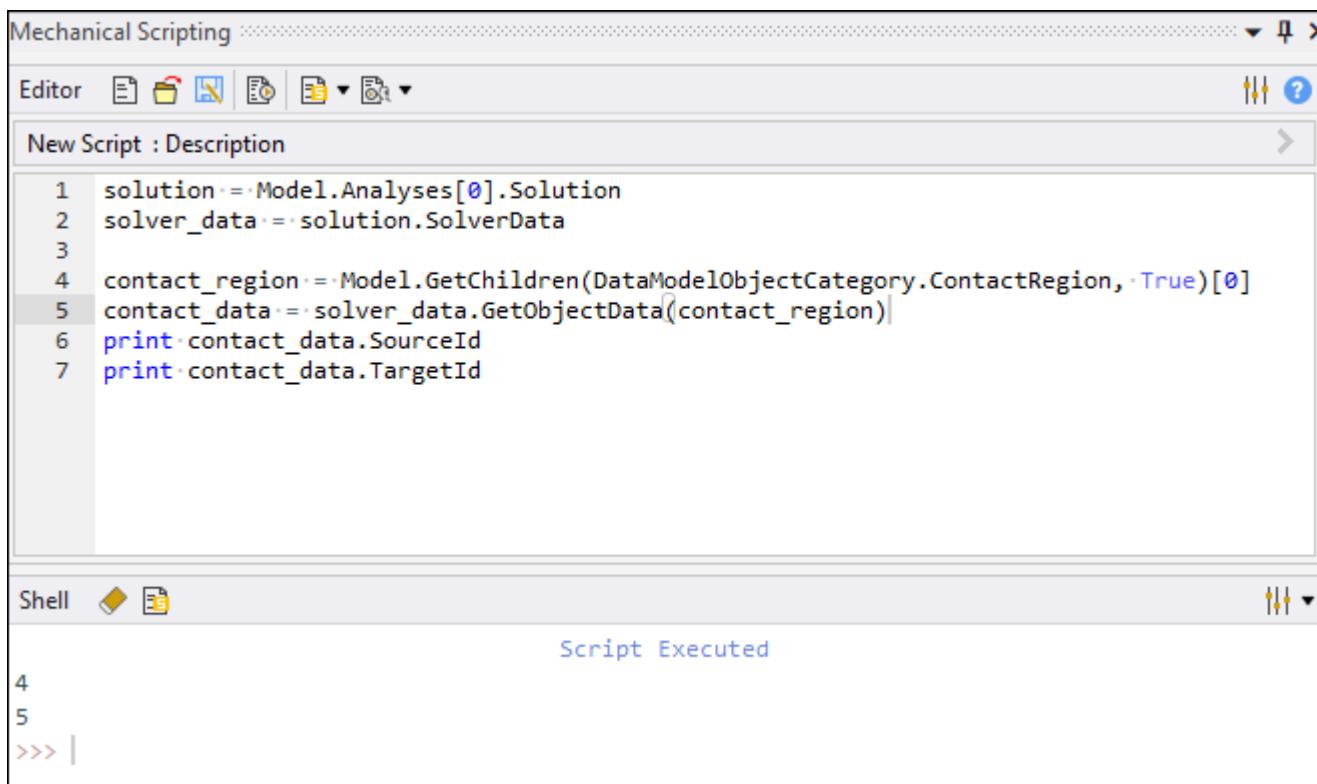
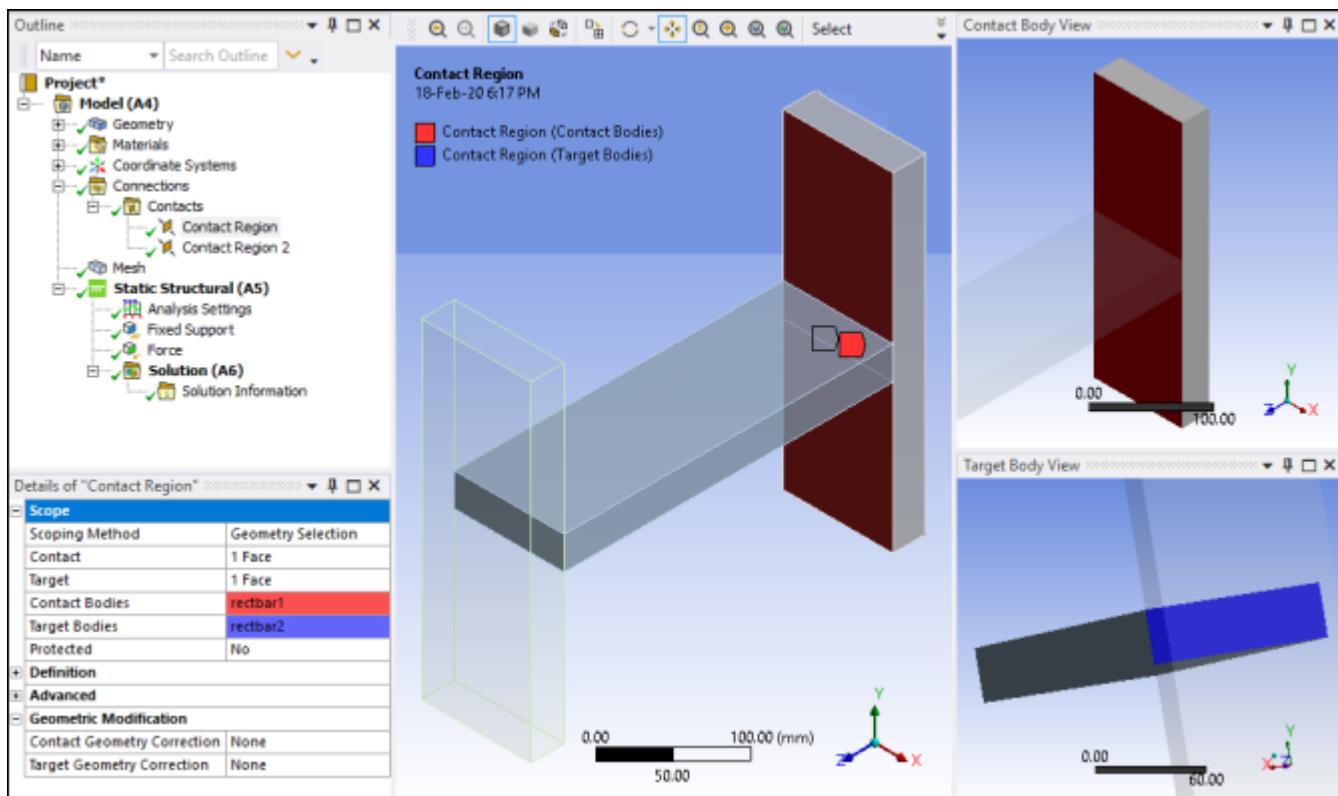
- **NodeIdsByMaterialId**
- **ElementIdsByMaterialId**

To use these functions with a Mechanical callback, such as **OnSolve**, **OnGenerate**, or **Evaluate**, the callback must be used with the tag: **InvokeUIThread=False**.

Getting Object-Specific Data

You can use **SolverData** APIs to get object-specific data. For example, you can use the **ContactRegion** object to get the source ID and target ID of the contact region:

```
contact_region = Model.GetChildren(DataModelObjectCategory.ContactRegion, True)[0]
contact_data = solver_data.GetObjectData(contact_region)
contact_data.SourceId
contact_data.TargetId
```



You can use **SolverData** APIs to retrieve object data for the following objects:

- AM Support

- Beam Connection
- Bearing
- Body
- Contact Region
- Coordinate System
- Joint
- Layered Section
- Pretension Bolt Load
- Remote Point
- Spring
- Surface Coating
- Surface Load
- Import Surface Loads ([Imported Load from External Data](#))

For a scripting example, see [Retrieve Object Details Using SolverData APIs \(p. 181\)](#).

Getting Object-Specific Data for Imported Objects

You can use **SolverData** APIs to get object-specific data for imported objects from [External Model](#). By using **GetObjectData** for imported objects, you get the collection of solver data information for all rows present in the imported object.

For example, you can use the imported coordinate system to get the collection of coordinate system IDs for all rows present:

```
solution = Model.Analyses[0].Solution
solver_data = solution.SolverData
imported_coordinate_system = DataModel.GetObjectsByType(DataModelObjectCategory.ImportedCoordinateSystems)[0]
imported_coordinate_system_data = solver_data.GetObjectData(imported_coordinate_system)
imported_coordinate_system_data[0].SystemId
imported_coordinate_system_data[1].SystemId
imported_coordinate_system_data[2].SystemId
```

Mechanical Scripting

Editor

New Script : Description

```
1 solution = Model.Analyses[0].Solution
2 solver_data = solution.SolverData
3
4 #ImportedCoordinateSystemData
5 imported_coordinate_system = DataModel.GetObjectsByType(DataModelObjectTypeCategory.ImportedCoordinateSystems)[0]
6 imported_coordinate_system_data = solver_data.GetObjectData(imported_coordinate_system)
7
8 print imported_coordinate_system_data[0].SystemId
9 print imported_coordinate_system_data[1].SystemId
10 print imported_coordinate_system_data[2].SystemId
11 print imported_coordinate_system_data[3].SystemId
12 print imported_coordinate_system_data[4].SystemId
```

Shell

Script Executed

```
13
14
15
16
17
>>> |
```

You can use `SolverData` APIs to retrieve object data for the following imported objects:

- Imported Bolt Pretensions

- Imported Contacts
- Imported Coordinate Systems
- Imported Flexible Remote Connectors
- Imported Pre-Meshed Bolt Pretensions
- Imported Rigid Remote Connectors
- Imported Spring Connectors
- Imported Surface Loads

A code example follows for each of these imported objects.

Imported Bolt Pretension

```
STAT_STRUC = Model.Analyses[0]
SOLN = STAT_STRUC.Solution
SOLVER_DATA = SOLN.SolverData
imported_bolt_pretension = DataModel.GetObjectsByType(DataModelObjectCategory.ImportedBoltPretensions)[0]
imported_bp_data = SOLVER_DATA.GetObjectData(imported_bolt_pretension)
imported_bp_data[0].PretensionNodeIds
imported_bp_data[0].RealConstantIds
```

Imported Contacts

```
STAT_STRUC = Model.Analyses[0]
SOLN = STAT_STRUC.Solution
SOLVER_DATA = SOLN.SolverData
imported_contacts = DataModel.GetObjectsByType(DataModelObjectCategory.ImportedContacts)[0]
imported_contacts_data = SOLVER_DATA.GetObjectData(imported_contacts)
imported_contacts_data[0].TargetId
imported_contacts_data[0].SourceId
```

Imported Coordinate System

```
STAT_STRUC = Model.Analyses[0]
SOLN = STAT_STRUC.Solution
SOLVER_DATA = SOLN.SolverData
imported_cs = DataModel.GetObjectsByType(DataModelObjectCategory.ImportedCoordinateSystems)[0]
imported_cs_data = SOLVER_DATA.GetObjectData(imported_cs)
imported_cs_data[0].SystemId
imported_cs_data[1].SystemId
imported_cs_data[2].SystemId
```

Imported Flexible Remote Connectors

```
STAT_STRUC = Model.Analyses[0]
SOLN = STAT_STRUC.Solution
SOLVER_DATA = SOLN.SolverData
imported_flexible_connector = DataModel.GetObjectsByType(DataModelObjectCategory.ImportedFlexibleRemoteConnectors)[0]
imported_frc_data = SOLVER_DATA.GetObjectData(imported_flexible_connector)
imported_frc_data[0].NodeId
```

Imported Pre-Meshed Bolt Pretension

```
STAT_STRUC = Model.Analyses[0]
SOLN = STAT_STRUC.Solution
```

```
SOLVER_DATA = SOLN.SolverData
imported_premeshed_bolt_pretension = DataModel.GetObjectsByType(DataModelObjectCategory.ImportedPremeshedBoltPre
imported_pbp_data = SOLVER_DATA.GetObjectData(imported_premeshed_bolt_pretension)
imported_pbp_data[0].PretensionNodeIds
imported_pbp_data[0].RealConstantIds
```

Imported Rigid Remote Connectors

```
STAT_STRUC = Model.Analyses[0]
SOLN = STAT_STRUC.Solution
SOLVER_DATA = SOLN.SolverData
imported_rigid_connector = DataModel.GetObjectsByType(DataModelObjectCategory.ImportedRigidRemoteConnectors)[0]
imported_rrc_data = SOLVER_DATA.GetObjectData(imported_rigid_connector)
imported_rrc_data[0].NodeId
imported_rrc_data[1].NodeId
imported_rrc_data[2].NodeId
```

Imported Spring Connectors

```
STAT_STRUC = Model.Analyses[0]
SOLN = STAT_STRUC.Solution
SOLVER_DATA = SOLN.SolverData
imported_spring_connector = DataModel.GetObjectsByType(DataModelObjectCategory.ImportedSpringConnectors)[0]
imported_sc_data = SOLVER_DATA.GetObjectData(imported_spring_connector)
imported_sc_data[0].RealConstantId
imported_sc_data[0].ElementId
imported_sc_data[1].RealConstantId
imported_sc_data[1].ElementId
```

Imported Surface Loads

```
STAT_STRUC = Model.Analyses[0]
SOLN = STAT_STRUC.Solution
SOLVER_DATA = SOLN.SolverData
Surface_Loads = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Surface Loads(External Model)'][0]
imported_sl_data = SOLVER_DATA.GetObjectData(Surface_Loads)
imported_sl_data[0].LoadTypes
imported_sl_data[1].LoadTypes
imported_sl_data[2].LoadTypes
imported_sl_data[0].GetSurfaceEffectElementTypeId(ExternalModel.ImportedSurfaceLoadType.Convection)
imported_sl_data[1].GetSurfaceEffectElementTypeId(ExternalModel.ImportedSurfaceLoadType.Pressure)
imported_sl_data[2].GetSurfaceEffectElementTypeId(ExternalModel.ImportedSurfaceLoadType.HeatFlux)
```


Boundary Conditions

This section describes APIs that enable you to manipulate boundary condition fields.

Unlike many other tree objects, boundary conditions are defined using both the **Details** view and tabular data. This is because boundary conditions can be time, space, or frequency dependent. As such, the APIs use a type of object called **Field**. Fields provide access to both independent and dependent variables defined in the tabular data, as well as the function that can be defined in the **Details** view.

The APIs for editing boundary conditions mirror the actions that can be performed manually in the Mechanical interface. With the API, you can:

- Set tabular data, which includes setting discrete input and output values
- Set variable definition types (tabular, formula, or free)

Input and Output Variables

Inputs and outputs are represented by objects of the type **Variable**. For example, a force can be defined with **time** as the input variable representing the time values in seconds across the time steps defined in the analysis settings and **magnitude** as the output variable representing the magnitude of the force at those time values.

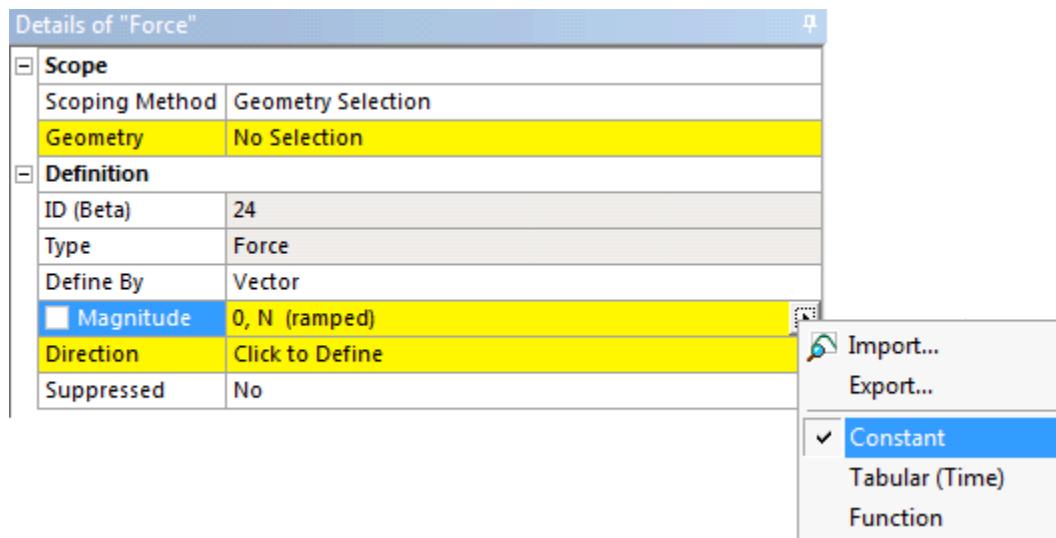
Variable Definition Types

A **Field** can be defined using of three **variable definition types**:

- **Discrete**. The variable contains a discontinuous set of values. By default, most variables are initially defined as discrete. Tabular or constant boundary conditions are considered discrete in the API.
- **Formula**. The variable is a continuous function depending on an expression, such as "**time*10**".
- **Free**. This variable definition type is available only for certain boundary conditions such as displacements.

Setting Input Loading Data Definitions

For input variables, each variable definition type corresponds to one or more of the input loading data definition options available in the **Details** view in Mechanical. You can determine the loading data definition options available for a given input by checking the flyout menu in the user interface.



Input loading data definitions are ranked in terms of complexity:

Complexity	Variable Definition Type	Input Loading Data Definition	Description
1	Discrete	Constant (stepped or ramped)	One value if stepped, two values if ramped.
2	Discrete	Tabular	Values listed in tabular format.
3	Formula	Function	Values generated by the expression assigned to the variable.
NA	Free	NA	Indicates that the boundary condition does not add a constraint for the specified degree of freedom (DOF). (A value of 0 indicates that the DOF is constrained.)

When you use the [Field API](#) to define a boundary condition field, it automatically opts for the least complex loading data definition that is applicable to the input. For example, if a given input could be defined as **Constant (ramped)** but you execute commands defining it as **Tabular**, the API defines the input as **Constant (ramped)**.

Setting Variable Definition Types

The property **DefinitionType** on the variable allows you to get or set the definition types of a field when used on the output variable.

Note:

You can also change the variable definition type to **Discrete** or **Free** by using the property **Variable.DiscreteValues** as described in [Setting Discrete Values for Variables \(p. 73\)](#).

An example follows, consisting of the following steps:

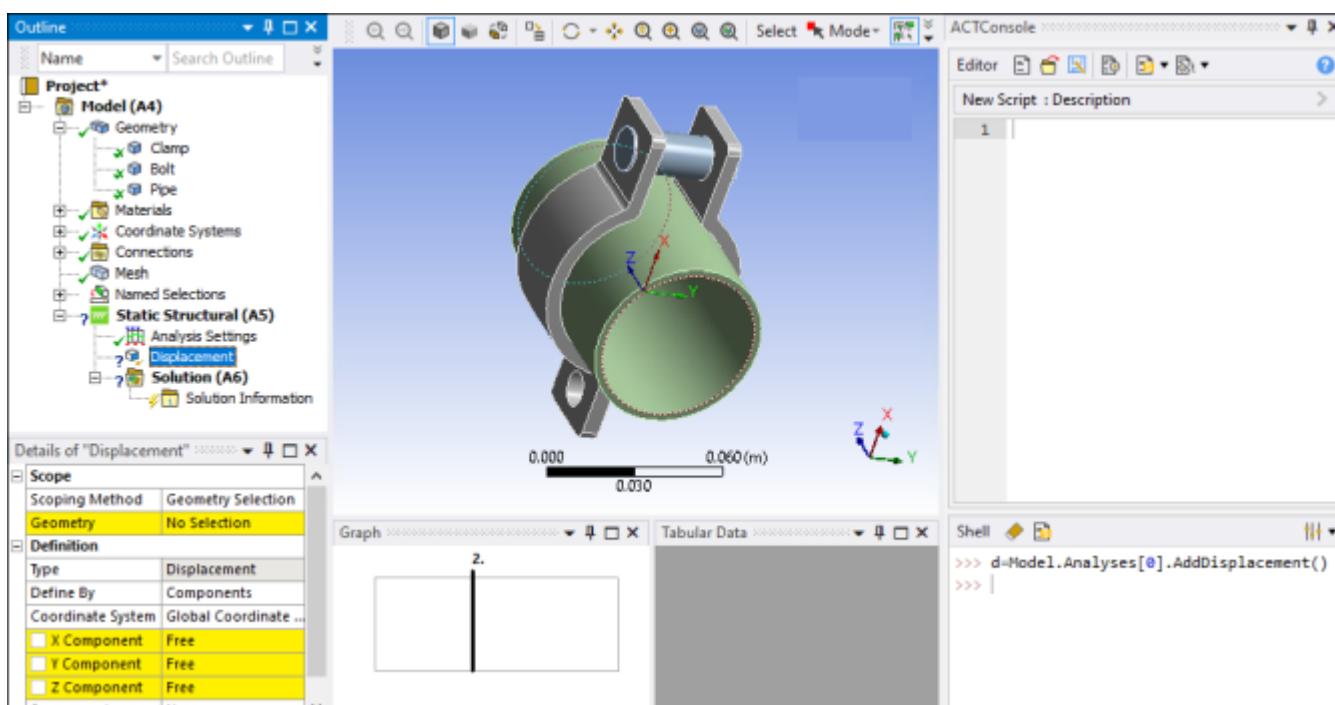
- Creating a Displacement and Verifying Its Variable Definition Types
- Changing the Y Component from Free to Discrete
- Changing the Y Component Back to Free

Creating a Displacement and Verifying Its Variable Definition Types

You start by creating a displacement:

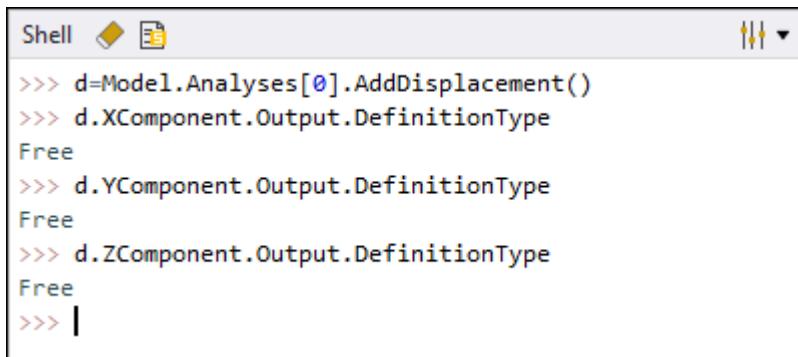
```
d=Model.Analyses[0].AddDisplacement()
```

In Mechanical, you can see that by default, the variable definition types for the displacement's X, Y, and Z components are set to **Free**.



You can verify this by executing the following commands one at a time.

```
d.XComponent.Output.DefinitionType  
d.YComponent.Output.DefinitionType  
d.ZComponent.Output.DefinitionType
```



```
Shell < > < >
>>> d=Model.Analyses[0].AddDisplacement()
>>> d.XComponent.Output.DefinitionType
Free
>>> d.YComponent.Output.DefinitionType
Free
>>> d.ZComponent.Output.DefinitionType
Free
>>> |
```

Note:

Even though the expression `d.XComponent.Output` would produce the same output in the console when the component is **Free**, be aware that output is an instance of the **Variable** class. This object is then converted into a string by the console. Appending `.DefinitionType` to the expression yields the actual enum value.

Changing the Y Component from Free to Discrete

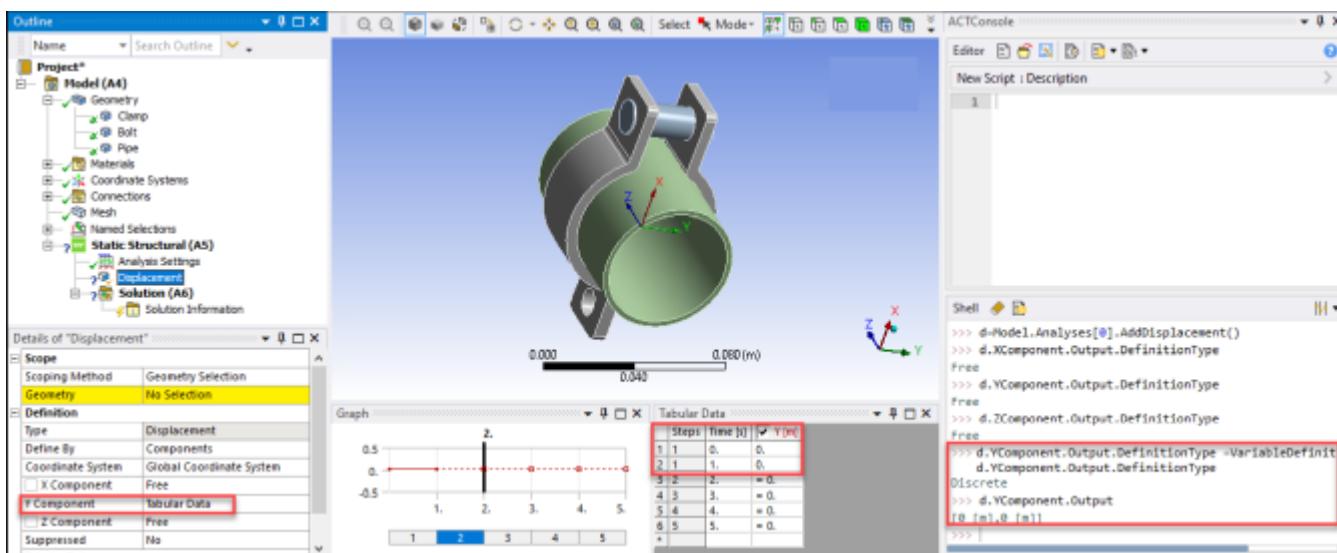
Next, you change the variable definition type for the Y component from **Free** to **Discrete** and then access the output values:

```
d.YComponent.Output.DefinitionType =VariableDefinitionType.Discrete
d.YComponent.Output.DefinitionType

d.YComponent.Output
```

In the figure that follows, you can see that:

- The **Y Component** is set to **Tabular Data** because tabular data is the least complex discrete loading data definition that is applicable to this input.
- The **Tabular Data** window is now populated with output values of **0** and **0**.
- In the **Shell**, you can verify that the variable definition type is set to **Discrete**.
- In the **Shell**, you can verify that the output values are **0** and **0**.

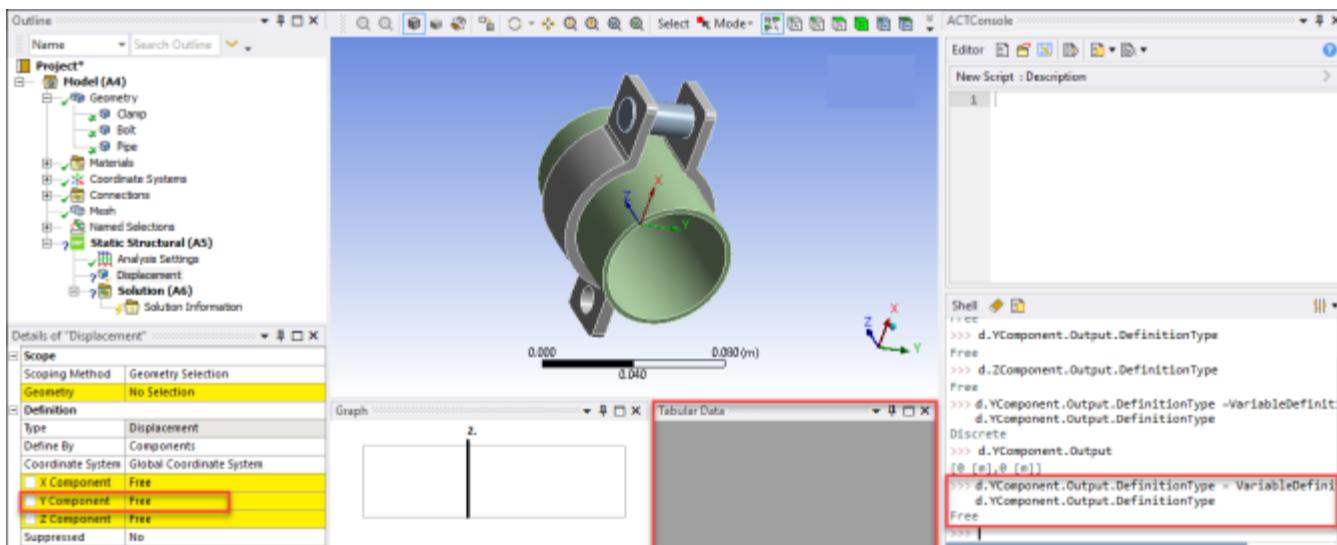


Changing the Y Component Back to Free

Finally, you change the variable definition type for the Y output from **Discrete** back to **Free**:

```
d.YComponent.Output.DefinitionType = VariableDefinitionType.Free
d.YComponent.Output.DefinitionType
```

In the following figure, you can see that the **Y Component** is set back to **Free** and the **Tabular Data** window is now empty.



Setting Discrete Values for Variables

The property **DiscreteValues** on the variable allows you to get and set the input and output variable values of a field:

[Controlling the Input Variable](#)

[Controlling the Output Variable](#)

Controlling the Input Variable

The list content provided to an input variable determines if it is set to a constant value or to **Tabular Data**. If the list contains only one quantity, the input is set to this constant value. If the list contains multiple quantities, the input is set to **Tabular Data**.

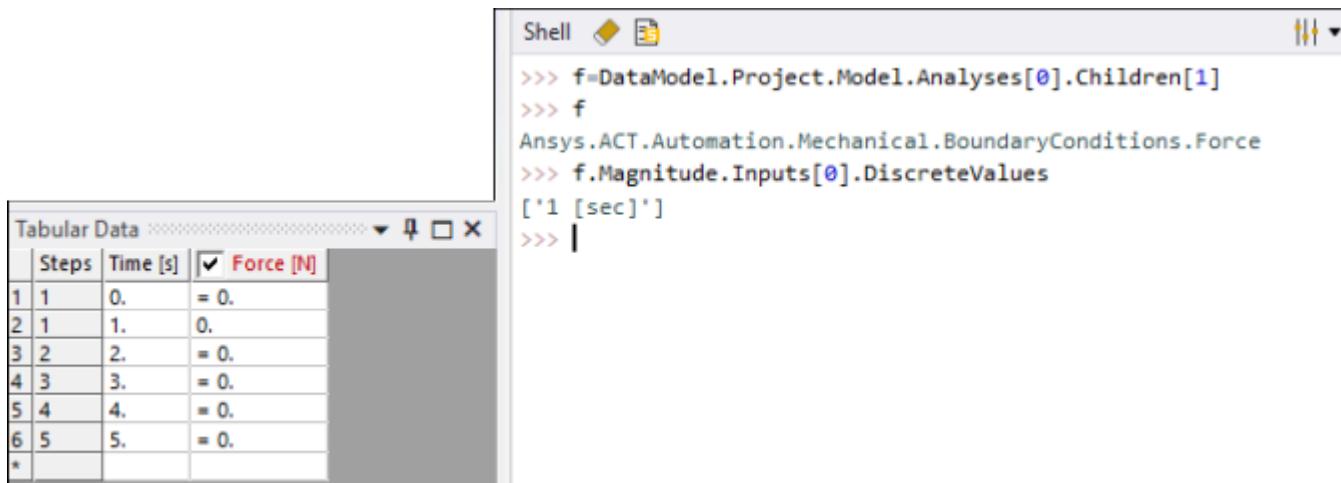
Note:

Input values for a discrete variable must be in a sorted order as shown in this example:

```
Pressure.Magnitude.Inputs[0].DiscreteValues = [Quantity("0 [s]"), Quantity("1 [s]"), Quantity("2 [s]"), Qua
```

Getting Discrete Input Values

You can get the discrete input values for a given input variable. For example, a force in a five-step analysis looks similar to the following figure by default. The six rows in the **Tabular Data** window correspond to the discrete values of the input variable, **time**, with **0** seconds as the start time of the first step and **5** seconds as the end time of the last step.



Setting Discrete Input Values

You can change discrete values for a given input variable. For example, the following code sample adds a discrete value, inserting a value of **0.5** seconds without defining an additional step in the analysis:

```
f.Magnitude.Inputs[0].DiscreteValues = [Quantity("0[sec]"),Quantity("0.5[sec]"),Quantity("1[sec]")]
```

In the figure that follows, you can see that:

- A new row exists for **0.5** seconds.
- The output cells for rows 3 through 7 have a yellow background. This is because values have not been set for the output variable.

Tabular Data			
	Steps	Time [s]	Force [N]
1	1	0.	0.
2	1	0.5	0.
3	1	1.	
4	2	2.	
5	3	3.	
6	4	4.	
7	5	5.	
*			

Removing Discrete Values

You can also remove a discrete value, deleting a row by defining a shorter list of values for the input variable. In the line of code that follows, the list specifies a single value, indicating that the other rows should be removed:

```
f.Magnitude.Inputs[0].DiscreteValues = [Quantity("0[sec]")]
```

In the **Tabular Data** window, only six rows now show once again. The value **=0** is actually a repeat of the value in the first row and does not correspond to an actual value stored for the variable. This is because the tabular data in Mechanical always displays a row for time values that correspond to step end times.

Tabular Data			
	Steps	Time [s]	Force [N]
1	1	0.	0.
2	1	1.	= 0.
3	2	2.	= 0.
4	3	3.	= 0.
5	4	4.	= 0.
6	5	5.	= 0.
*			

Controlling the Output Variable

Output variables are not limited to the **Discrete** variable definition type, so the behavior when getting and setting outputs is slightly different from that of inputs.

Getting Discrete Output Values

The following table shows the **get** behavior for the different variable definition types.

Variable Definition Type	Behavior
Discrete	Same as described earlier for getting discrete input values.
Free	For C#, the property returns null . For Python, the property returns None .
Formula	The property returns the series of evaluated variable values.

Setting Discrete Output Values

The following table shows the **set** behavior according to the list content provided to the property.

List Content	Behavior
None or null	The variable is switched to the Free definition type if it is supported by the boundary condition. Otherwise, an error is issued.
Single quantity object	The variable is switched to the Constant definition type if supported in the interface. Otherwise, Constant (ramped) can be chosen. If neither of those two types are supported, Tabular Data is the default.
Two quantity objects, with the first value being 0	The variable is switched to Constant (ramped) if supported in the interface. Otherwise, Tabular Data is the default.
As many quantity objects as discrete values for the input variables	The variable is switched to Tabular Data .
Does not fall into any of the above categories	An error is issued, indicating that you should either: <ul style="list-style-type: none"> Provide the appropriate number of values. Change the number of input values first, as the input value count is used to determine the row count of the tabular data. Output values must then comply with this number of rows. The following figure shows a sample message. <div style="border: 1px solid black; padding: 5px;"> <p>> f.Magnitude.Output.DiscreteValues = [Quantity('1 [N]'), Quantity('10 [N]'), Quantity('5 [N]')]</p> <p>✖ Expected 2 values but 3 were found. Consider changing the variable's input discrete values. Parameter name: value</p> </div>

Setting Variable Definition Type by Acting on Discrete Values

The variable definition type can also be modified by setting the **DiscreteValues** property on the variable. To set a variable to a particular variable type, you set the values that are consistent with that variable type. Examples follow:

- To set an input variable to **Tabular Data**, assign a list of discrete values:

```
d.XComponent.Output.DiscreteValues = [Quantity("1 [m]"), Quantity("10 [m]")]
```

- To set a variable to **Free**, assign **None** to its **DiscreteValues**:

```
d.XComponent.Output.DiscreteValues = None
```

Adding a Load

You use the object [Analysis](#) to add a load. This topic describes adding loads to a static structural analysis.

To access the first analysis in the Mechanical project:

```
static_structural = Model.Analyses[0]
```

To change the number of steps of the analysis:

```
analysis_settings = static_structural.AnalysisSettings.NumberOfSteps = 4
```

To add and define a bolt and fixed support:

```
bolt = static_structural.AddBoltPretension()
bolt_scoping = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
bolt_scoping.Ids = [200]
bolt.Location = bolt_scoping
bolt.SetDefineBy(1, BoltLoadDefineBy.Load) # Change definition for step #1.
bolt.Preload.Output.SetDiscreteValue(0, Quantity("15 [N]")) # Change preload value for step #1.

support = static_structural.AddFixedSupport()
support_scoping = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
support_scoping.Ids = [104]
support.Location = support_scoping
```

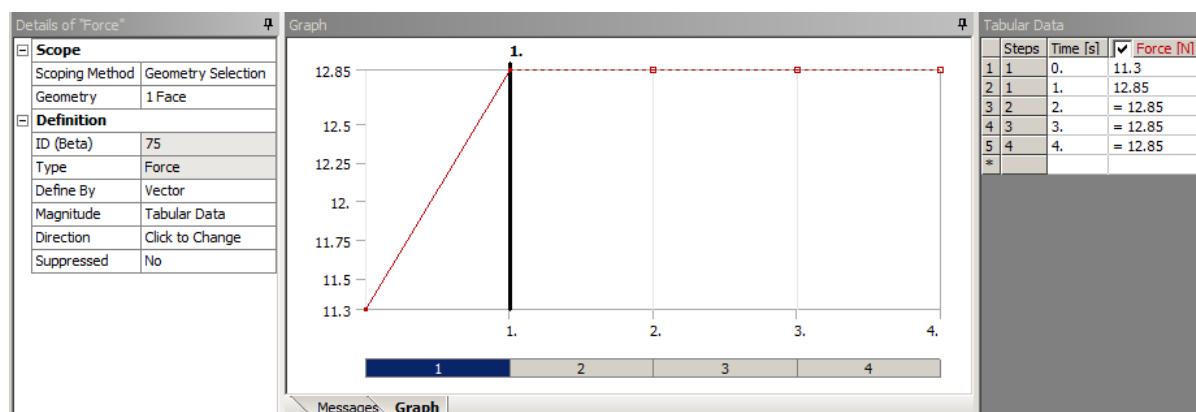
To add and define the external and internal pressures exerted on the pipe and the force on a section of the pipe:

```
pressure = static_structural.AddPressure()
pressure_scoping = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
pressure_scoping.Ids = [220]
pressure.Location = pressure_scoping
pressure.Magnitude.Output.Formula = '10*time'

pressure = static_structural.AddPressure()
pressure_scoping = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
pressure_scoping.Ids = [221]
pressure.Location = pressure_scoping
pressure.Magnitude.Output.DiscreteValues=[Quantity('6 [Pa]')]

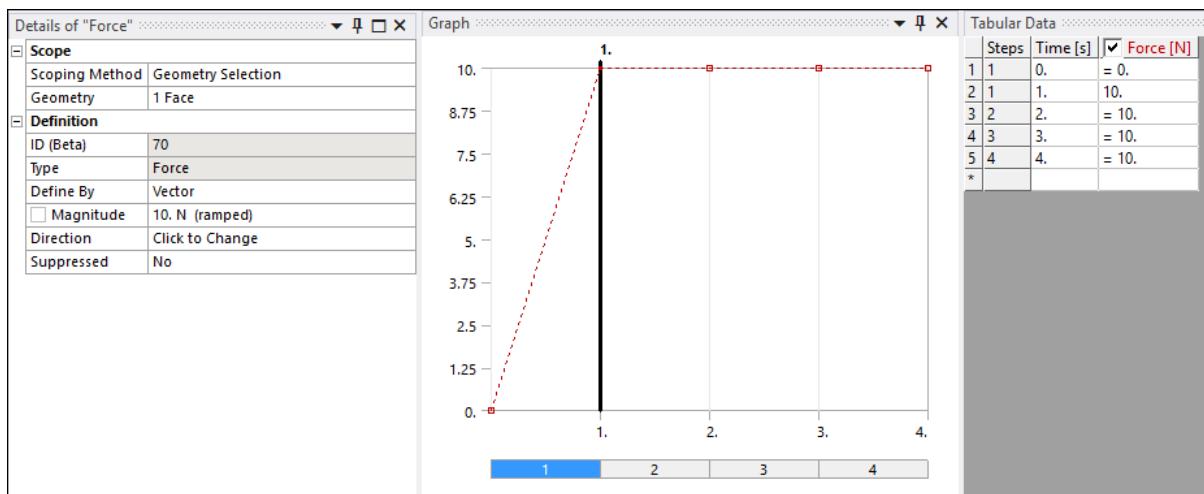
force = static_structural.AddForce()
force_scoping = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
force_scoping.Ids = [219]
force.Location = force_scoping
force.Magnitude.Output.DiscreteValues=[Quantity('11.3 [N]'), Quantity('12.85 [N]')]
```

Script execution results in the following force definition:



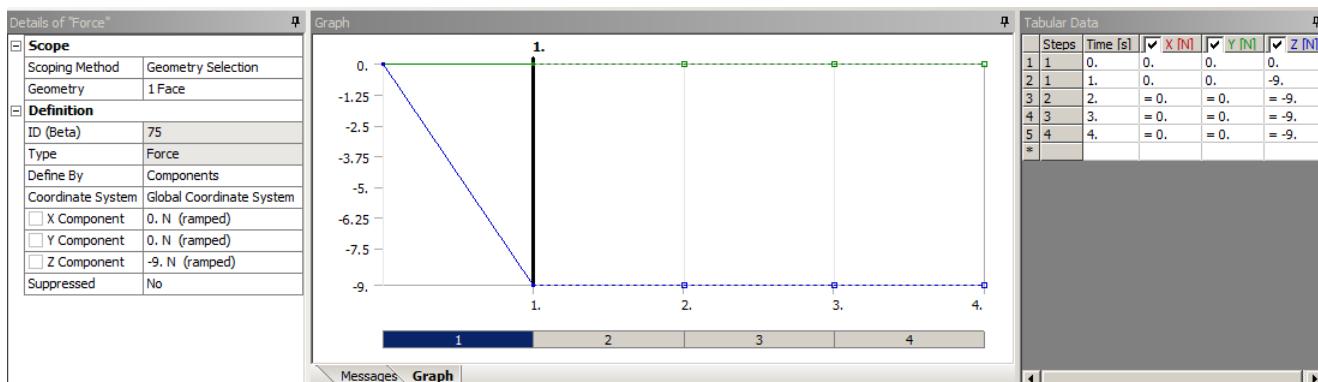
To define a constant value of the force:

```
force.Magnitude.Output.DiscreteValues=[Quantity('10 [N]')]
```



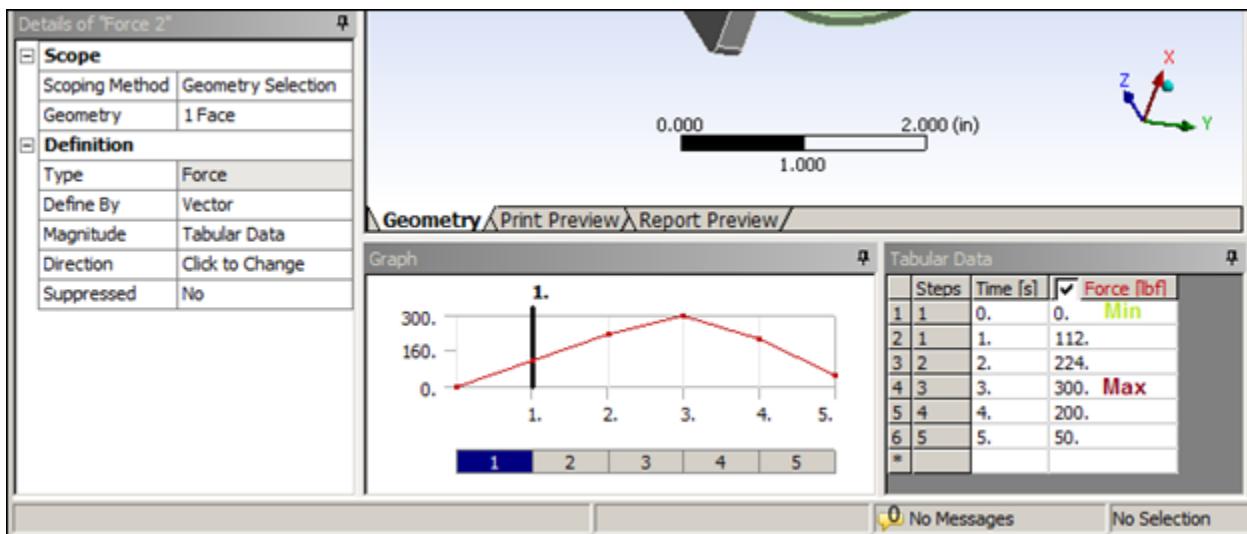
You can also change the force to be defined by components instead of by magnitude and set the values of an individual component:

```
force.DefineBy = LoadDefineBy.Components
force.ZComponent.Output.DiscreteValues = [Quantity('0 [N]'),Quantity('-9 [N]')]
```



Extracting Min-Max Tabular Data for a Boundary Condition

The [Field API](#) can be used to extract minimum and maximum tabular data for boundary conditions. For example, you can extract minimum and maximum applied force. The following figure displays tabular data for a force and highlights its minimum and maximum quantities.



First, get the project model object, analysis object, force object, and the output variable with the applied force variables:

```
mymodel = DataModel.Project.Model
anal = mymodel.Analyses[0]
f2 = anal.Children[2]
f2td = f2.Magnitude.Output
```

Next, get the tuple containing the minimum and maximum force quantities and then display these quantities by entering the variable name of the tuple:

```
mnmxf2 = f2td.MinMaxDiscreteValues
mnmxf2
```

Given the above tabular data, the following results display:

(0 [lbf], 300 [lbf])

The variable `mnmxf2` is a tuple (pair) of quantities. Each element in the tuple can be gotten by using the tuple's properties `Item1` and `Item2`.

To get and display only the minimum force quantity:

```
mnv = mnmxf2.Item1
mnv
```

Given the above tabular data, the following results display:

(0 [lbf])

To get and display only the maximum force quantity:

```
mxv = mnmxf2.Item2
mxv
```

Given the above tabular data, the following results display:

(300 [1bf])

Setting the Direction of a Boundary Condition

You can set the direction of a boundary condition.

To set the direction of a load to a known value:

```
pressure.Direction = Vector3D(1,0,0)
```

To set the direction of a load using the direction from a face or edge or between points:

```
sels = ExtAPI.SelectionManager.CurrentSelection
reversed = True
vec = Ansys.Mechanical.Selection.SelectionHelper.CreateVector3D(sels, reversed)
acceleration.Direction = vec
```

Worksheets

You can use Mechanical APIs to access or modify data that is displayed by the user interface in the worksheet panel:

[Named Selection Worksheet](#)

[Mesh Order Worksheet](#)

[Layered Section Worksheet](#)

[Bushing Joint Worksheet](#)

[Condensed Part Worksheet](#)

Named Selection Worksheet

Mechanical APIs support all available actions for named selection worksheets, as described in [Specifying Named Selections Using Worksheet Criteria](#) in the *Ansys Mechanical User's Guide*.

The following topics describe how to use APIs to define a named selection worksheet:

[Defining the Named Selection Worksheet](#)

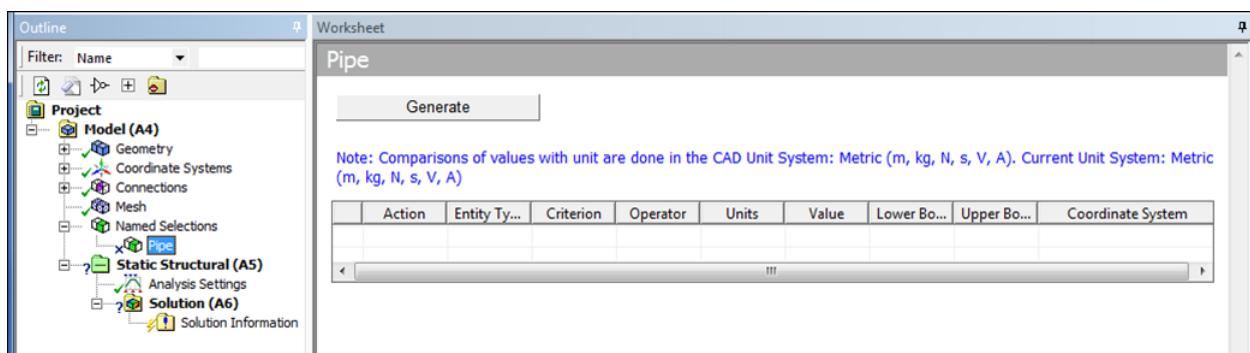
[Adding New Criteria to the Named Selection Worksheet](#)

Defining the Named Selection Worksheet

To define a named selection worksheet, you execute these commands:

```
sel = Model.AddNamedSelection()
sel.ScopingMethod=GeometryDefineByType.Worksheet
sel.Name = "Pipe"
pipews = sel.GenerationCriteria
```

This sample code creates the named selection object, renames it to **Pipe**, changes the property **Define By** to **Worksheet**, and stores a variable of the **NamedSelectionCriteria** object that can be used to manipulate the data in the worksheet.



Adding New Criteria to the Named Selection Worksheet

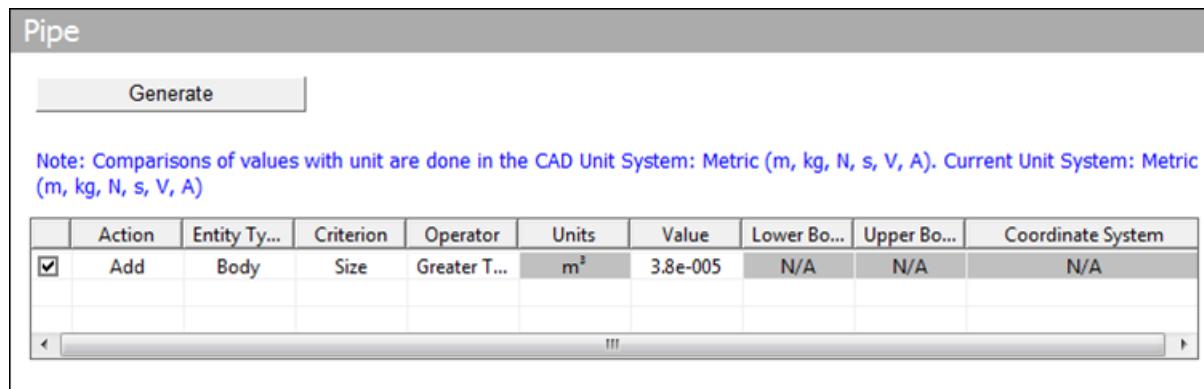
The named selection worksheet defines the criteria from which to generate a selection. Each row of the worksheet adds a new criteria and defines the relationship with the previous rows. The focus here is on adding a new row.

The **NamedSelectionCriteria** object behaves like a list and has standard list properties and methods including **Add()**, **Clear()**, **Count**, **Insert**, **Remove**, and a **[]** indexer. You can use the **Add()** method to add a **NamedSelectionCriterion**, which is the object for each element in the list and controls the data of a single row in the worksheet.

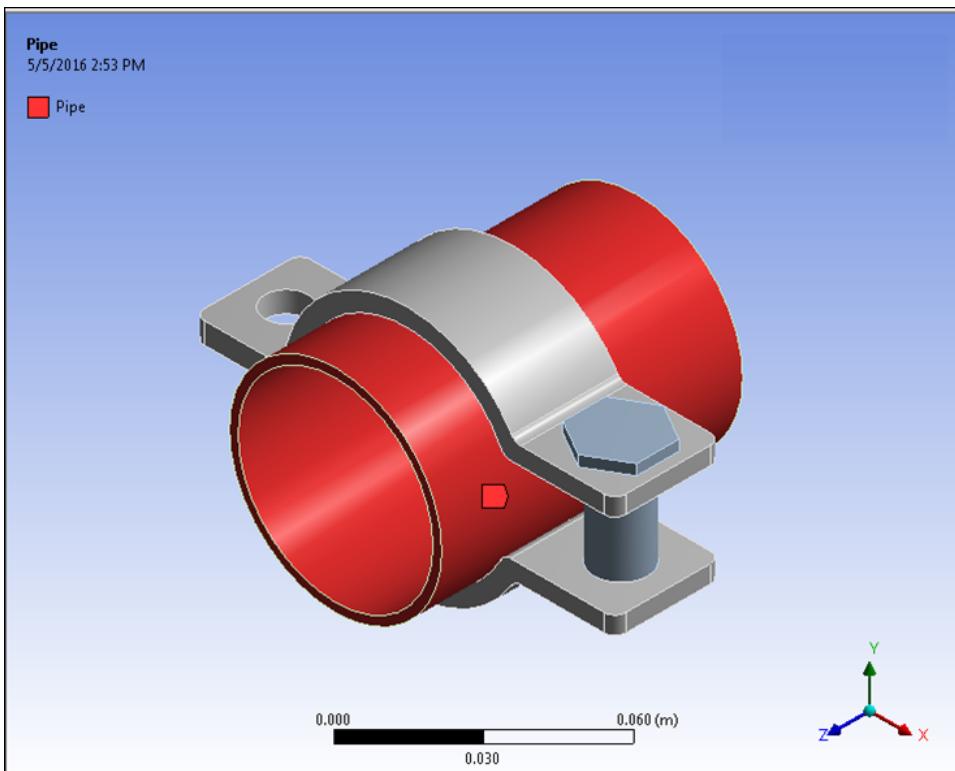
To add the first criterion to your worksheet:

```
pipews.Add(None)
pipews[0].EntityType=SelectionType.GeoBody
pipews[0].Criterion=SelectionCriterionType.Size
pipews[0].Operator=SelectionOperatorType.GreaterThan
pipews[0].Value=Quantity("3.8e-5 [m m m]")
sel.Generate()
```

This example defines a criterion to select bodies with a size value greater than **3.8e-5**.



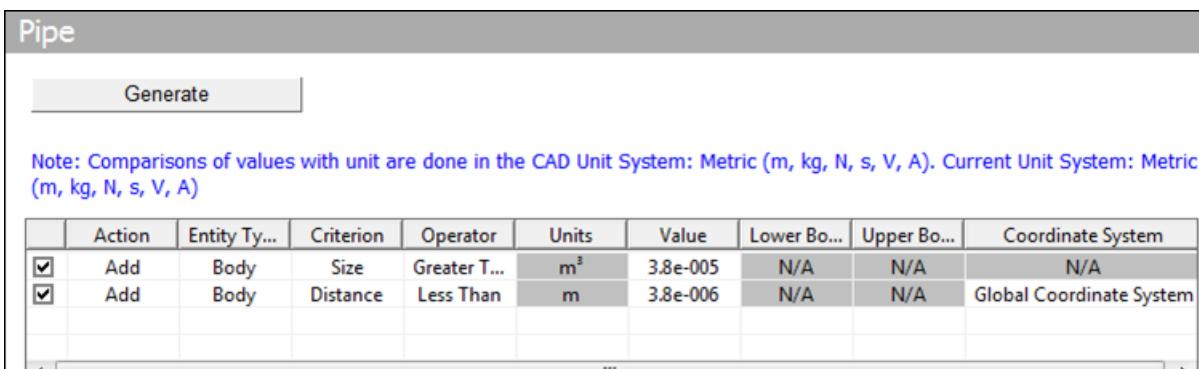
The method **Generate** on the **NamedSelection** object generates the selection based on the criteria. After executing this method, observe that the pipe body is selected in the **Graphics** view. In the **Details** view, the property **Total Selection** is set to **1 Body**.



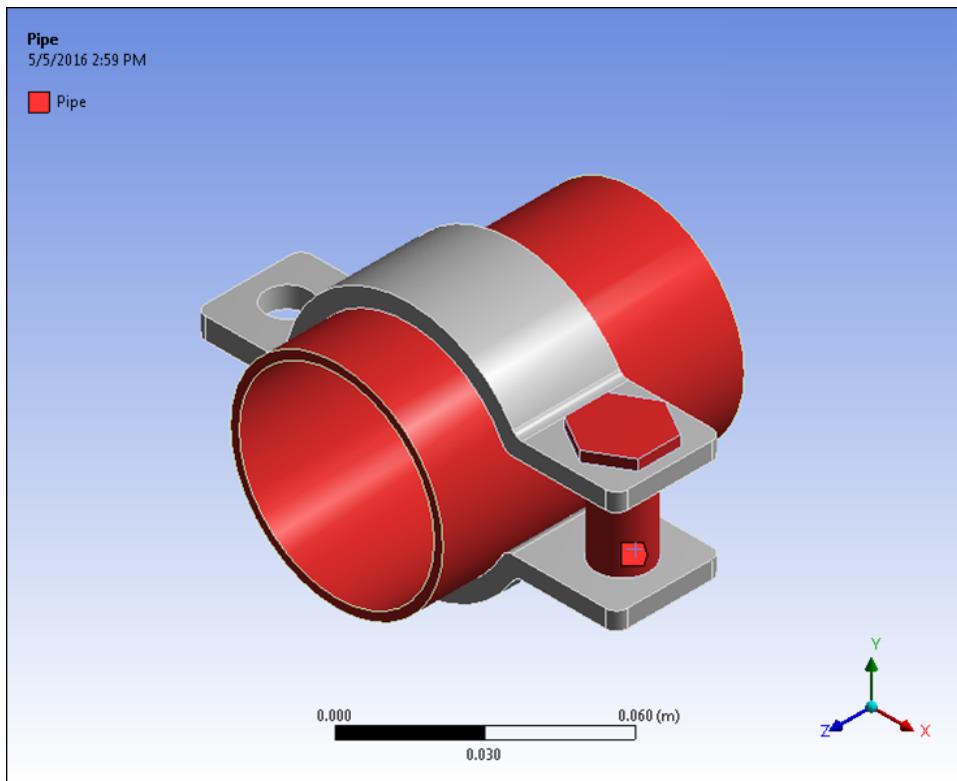
Next, you can add another criteria to include other bodies:

```
pipew[1].EntityType=SelectionType.GeoBody
pipew[1].Criterion=SelectionCriterionType.Distance
pipew[1].Operator=SelectionOperatorType.LessThan
pipew[1].Value=Quantity("3.8e-6 [m]")
sel.Generate()
```

This new criteria is defined to select bodies whose distance from a given coordinate system is less than **3.8e-6**. If you do not specify a coordinate system, it defaults to the global coordinate system.



After executing the **Generate** method, observe that the bolt body is now selected in the **Graphics** view, along with the pipe body. In the **Details** view, the property **Total Selection** is set to **2 Bodies**.

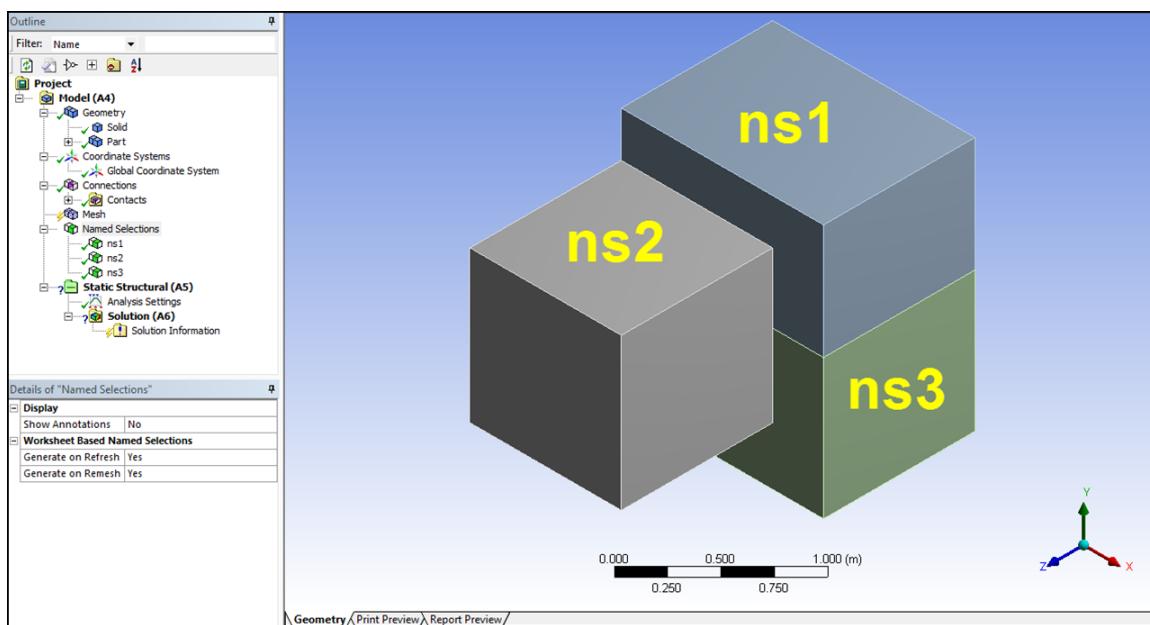


Mesh Order Worksheet

You can use Mechanical APIs to specify the order in which meshing steps are performed in Mechanical. This feature is described in [Using the Mesh Worksheet to Create a Selective Meshing History](#) in the *Ansys Meshing User's Guide*. The following restrictions apply to these APIs:

- The named selections must have **Entity Type** set to **Body**. Other entity types are not supported.
- The **Start Recording** and **Stop Recording** buttons do not have APIs.

The examples in this section assume that you have already defined two or more named selections in Mechanical. In the following figure, three named selections are defined.



Defining the Mesh Worksheet

To begin using the APIs to define the Mesh Worksheet as described in the following example, use the following variables to refer to the mesh worksheet data and the three named selections to be used in the worksheet. The [MeshControlWorksheet](#) object is used to control the worksheet data.

```
mws = Model.Mesh.Worksheet
nsels = Model.NamedSelections
ns1 = nsels.Children[0]
ns2 = nsels.Children[1]
ns3 = nsels.Children[2]
```

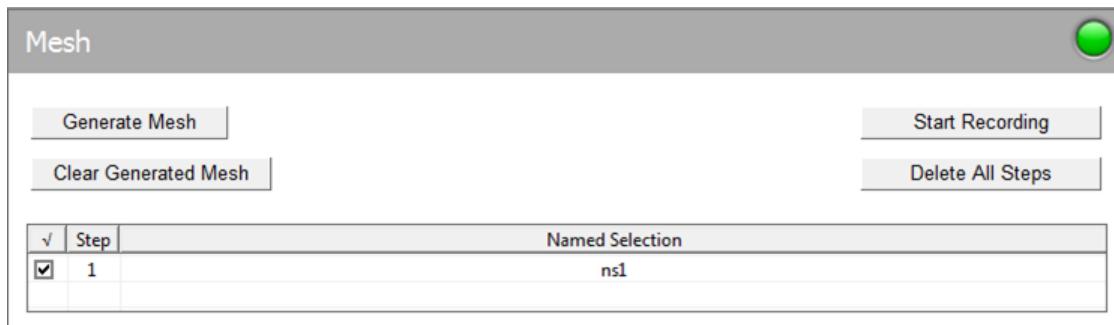
Adding New Rows in the Mesh Worksheet

To add the first row to your worksheet, you execute these commands:

```
mws.AddRow()
mws.SetNamedSelection(0,ns1)
mws.SetActiveState(0,True)
```

- The command **AddRow** adds an empty row.
- The command **SetNamedSelection** sets the value in the **Named Selection** column to your named selection variable **ns1**.
- The command **SetActiveState** sets **Active State** at the row index. This is indicated by the check mark in the left column.

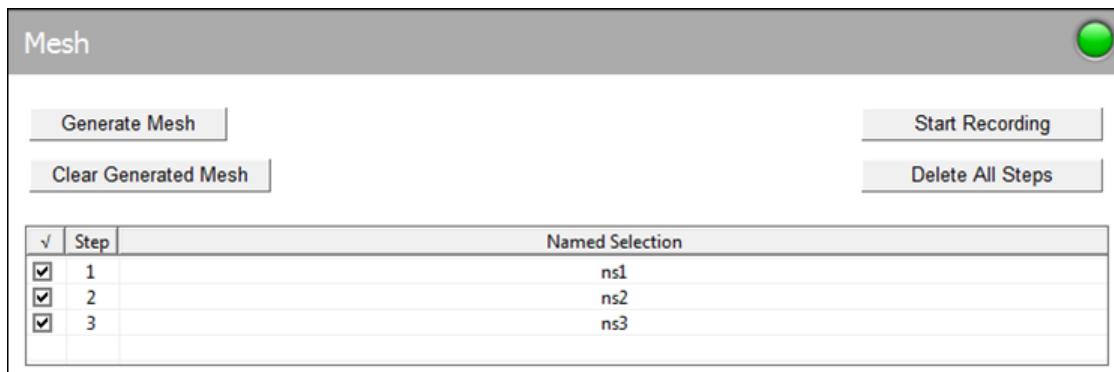
The following figure shows the mesh worksheet after the execution of these commands.



To add a row for each of the other two named selections, **ns2** and **ns3**, use these commands:

```
mws.AddRow()
mws.SetNamedSelection(1,ns2)
mws.SetActiveState(1,True)
mws.AddRow()
mws.SetNamedSelection(2,ns3.)
mws.SetActiveState(2,True)
```

The following figure shows the mesh worksheet after the two new rows have been added.

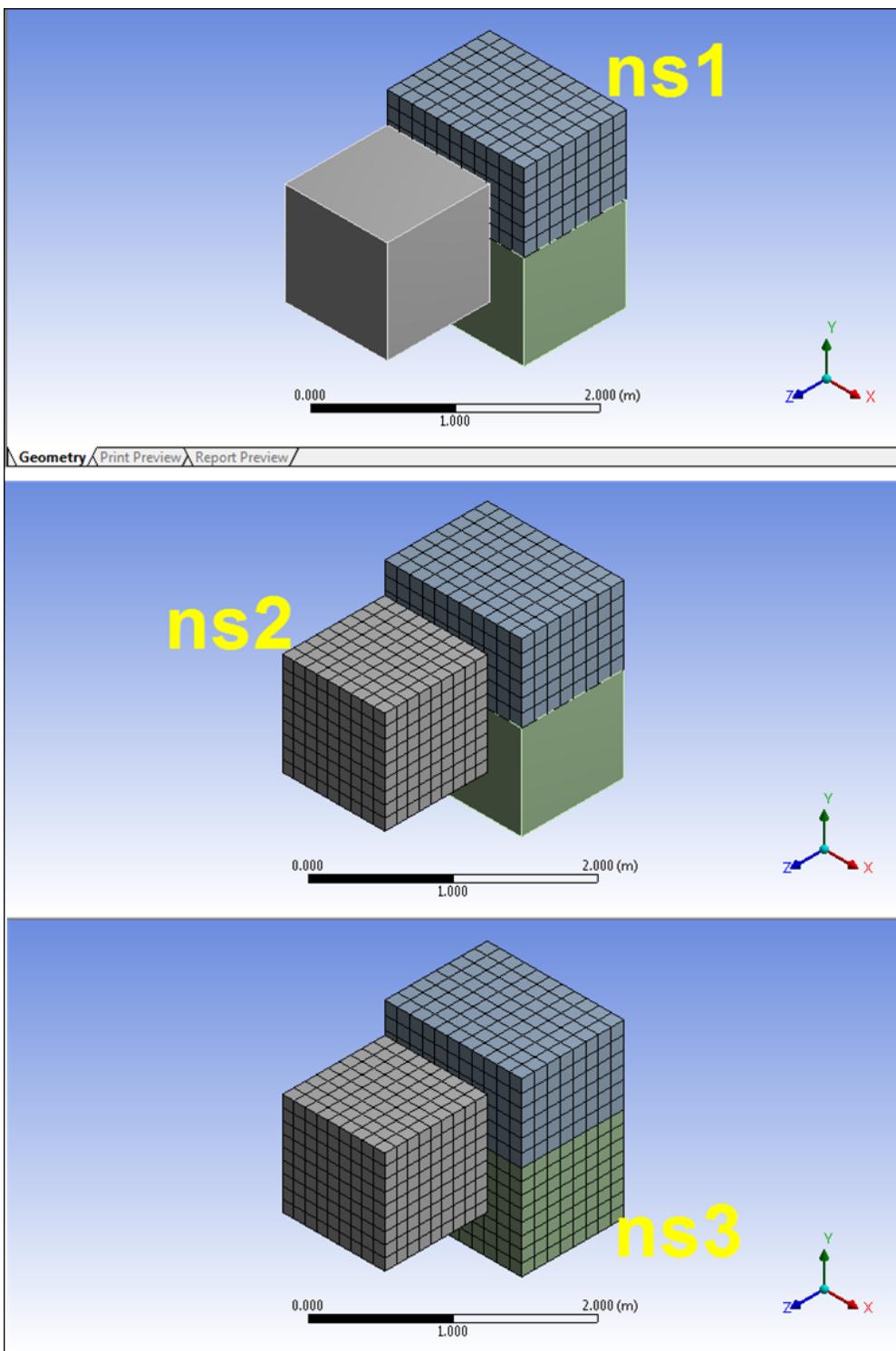


Meshing the Named Selections

When you check the **Graphics** tab, you can see that no mesh has been generated yet. To generate the mesh for the named selections in the mesh worksheet, execute the following command:

```
mws.GenerateMesh()
```

When you execute this command, the steps are processed one by one in the order specified by the worksheet. For each step, the bodies identified by the named selection are meshed using the meshing controls applied to them. By watching the mesh generation in the **Graphics** tab, you can see the order in which each body is meshed.



Working with Weld Worksheet

Weld Worksheet allows you to scope weld controls for the weld curves. **Weld Worksheet** scripting allows you to record the worksheet editing for automation. It enables you to select and edit multiple cells in a column in the worksheet.

To activate **Weld Worksheet**:

```
weld.UseWorksheet = YesNoType.Yes #Activates the worksheet.
```

To search for the **Weld** control with the given text:

```
weld.NameFilter = '' #Enter the text to search for inside the ''.  
weld.WeldWorksheetCreateControlForCurveBodies() #Creates weld control for the curve bodies.
```

To export the weld worksheet:

```
weld.WeldWorksheetExport("FilePath") #Exports the worksheet entries as .csv file and save it to the given file p
```

To import the weld worksheet:

```
weld.WeldWorksheetImport("FilePath") #Import the worksheet parameters as .csv file from the given file path.
```

To set the **Edge Mesh Size** in the worksheet:

```
row_id =[2] #Add the row id. Here, the row id is 2. Multiple row id can be added with comma separation.  
edgemesh_size=2.0 #Define the size of the edge mesh.  
weld.SetWeldWorksheetEdgeMeshSize([row_id], edgemesh_size)
```

To set the **Weld Angle** in the worksheet:

```
row_id =[1] #Add the row id. Here, the row id is 2. Multiple row id can be added with comma separation.  
weld_angle=30 #Define the weld angle.  
weld.SetWeldWorksheetWeldAngle([row_id], weld_angle)
```

To set the **Offset Layer Height** in the worksheet:

```
row_id = [1] #Add the row id. Multiple row id can be added with comma separation.  
offset_layer_height = 30 #Define offset layer height.  
weld.SetWeldWorksheetOffsetLayerHeight([row_id], offset_layer_height)
```

To set the **Number of Layers** in the worksheet:

```
row_id = [1] #Specify the row id. Multiple row id can be added with comma separation.  
num_layers = 1 #Define the required number of layers.  
weld.SetWeldWorksheetNumLayers([row_id], num_layers)
```

To set the **Offset Layer Growth Rate** in the worksheet:

```
row_id = [1] #Specify the row id. Multiple row id can be added with comma separation.  
offset_layer_growth_rate = 1.3 #Define the growth rate for offset layer  
weld.SetWeldWorksheetOffsetLayerGrowthRate([row_id], offset_layer_growth_rate)
```

To set the **Sharp Angle** in the worksheet:

```
row_id = [1] #Specify the row id. Multiple row id can be added with comma separation.  
sharp_angle = 20 #Define the sharp angle.  
weld.SetWeldWorksheetSharpAngle([row_id], sharp_angle)
```

To set the **Absolute Tolerance** in the worksheet:

```
row_id = [1] #Specify the row id. Multiple row id can be added with comma separation.
absolute_tolerance= 10 #Define the absolute tolerance.
weld.SetWeldWorksheetAbsTol([row_id], absolute_tolerance)
```

To set the **Lap Angle Tolerance** in the worksheet:

```
row_id = [1] #Specify the row id. Multiple row id can be added with comma separation.
lapangle_tolerance= 10 #Define the lap angle tolerance.
weld. SetWeldWorksheetLapAngleTol([row_id], lapangle_tolerance)
```

To set the **Material Id** in the worksheet:

```
row_id = [1] #Specify the row id. Multiple row id can be added with comma separation.
material_id= 10 #Specify the material id.
weld.SetWeldWorksheetMaterialId([row_id], material_id)
```

To set the **Smoothing** in the worksheet:

```
row_id = [10] #Specify the row id. Multiple row id can be added with comma separation.
smoothing= 0 #Set Smoothing to No. 0 is the default value. 1 - Enables Smoothing and 0 - Disables Smoothing.
weld.SetWeldWorksheetSmoothing([row_id], smoothing)
```

To enable the **Generated End-Caps** in the worksheet:

```
row_id = [3] #Specify the row id. Multiple row id can be added with comma separation.
generate_endcaps= 0 #Generates triangular end caps at the free ends of the weld.
weld.SetWeldWorksheetGenerateEndCaps(row_id, generate_endcaps)
```

Layered Section Worksheet

To get to the [LayeredSectionWorksheet](#) object:

```
lsws = Model.Geometry.Children[1].Layers
```

This object uses 0-based indices to refer to rows.

- To control the data in the material column, use the **GetMaterial** or **SetMaterial** method.
- To control the data in the **Thickness** column, use the **GetThickness** or **SetThickness** method.
- To control the data in the **Angle** column, use the **GetAngle** or **SetAngle** method.

Bushing Joint Worksheet

A bushing joint worksheet defines stiffness and damping coefficients via symmetric matrices. The matrices are fixed size and symmetric fixed, which means that the API cannot add or remove entries or modify the upper right triangle. The following topics describe how to use APIs for the bushing joint worksheet:

Getting the Bushing Joint Worksheet and Its Properties

Getting the Value for a Given Component

Getting the Bushing Joint Worksheet and Its Properties

To begin using the APIs to define the bushing joint worksheet as described in the following example, use the following variable to refer to the bushing joint worksheet data. It will be an instance of the [JointBushingWorksheet](#) object.

```
bws = Model.Connections.Children[1].Children[0].BushingsWorksheet
```

Getting the Value for a Given Component

The API uses 0-based row indices using a method for each column. Because only the lower left triangle can be controlled, each method will have a different range of valid indices.

For example, to get the coefficient for the stiffness per unit Y at row index 1:

```
bws.GetBushingsStiffnessPerUnitY(1)
```

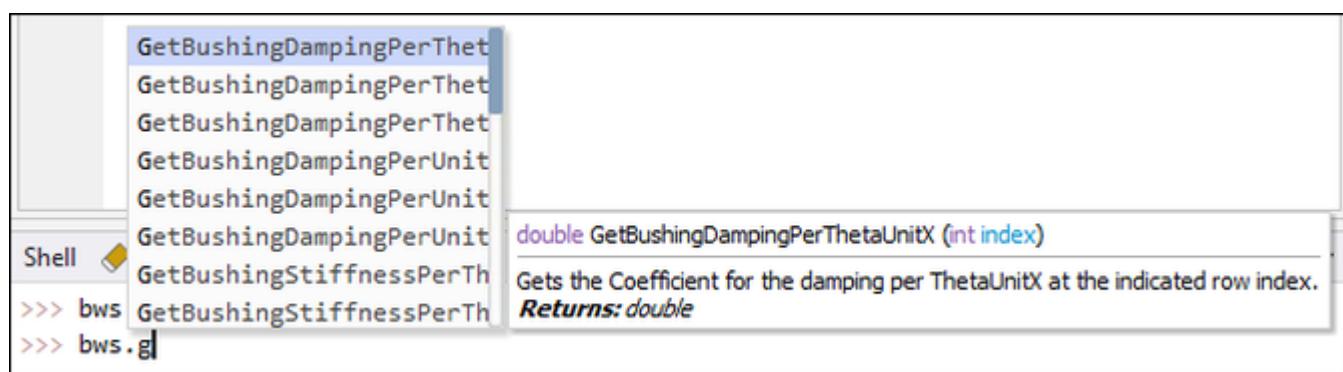
Assume that **136073.550978543** displays as the stiffness per unit Y.

Now, assume you have entered the following:

```
bws.GetBushingsStiffnessPerUnitZ(0)
```

The given index of 0 (zero) produces an error message because the cell indicated is in the upper right triangle. As per the error message, the valid range of indices is 2 to 5, inclusive.

The following figure shows some of the many methods available for getting coefficients for damping and stiffness using the Mechanical **Scripting** view's autocomplete feature.



Condensed Part Worksheet

Mechanical APIs support all available actions for the condensed part worksheet.

Getting the Condensed Part Worksheet Object

To get the condensed part worksheet object, execute these commands:

```
cp = DataModel.GetObjectsByType(DataModelObjectCategory.CondensedPart)[0]
cpws = cp.Interfaces
```

Because the index is 0, this sample code gets the first condensed part and stores a variable of the **CondensedPartInterfaces** object that can be used to modify the data in the worksheet.

Adding New Interfaces to the Condensed Part Worksheet

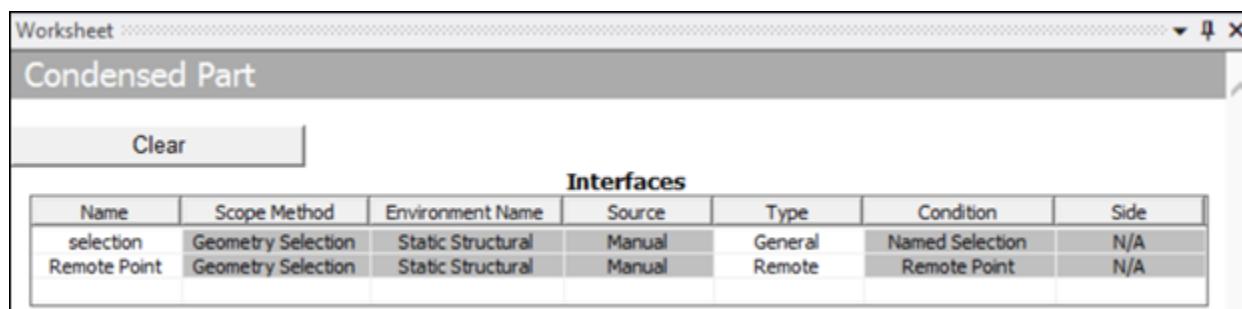
The **CondensedPartInterfaces** object has these methods:

- **Add()**
- **Clear()**
- **RemoveAt()**
- **Indexer**

You can use the **Add()** method to add **CondensedPartInterface**, which is the object for each element in the list and controls the data of a single row in the worksheet.

For example, to add **CondensedPartInterface** to the worksheet:

```
cpws.Add()
cpws[0].Type = CondensedPartInterfaceType.General
ns = DataMode.GetObjectsByType(DataModelObjectCategory.NamedSelection)[0]
cpws[0].NamedSelection = ns
cpws.Add()
cpws[1].Type = CondensedPartInterfaceType.General
rp= DataMode.GetObjectsByType(DataModelObjectCategory.RemotePoint)[0]
cpws[1].NamedSelection = rp
```



Graphics

This section describes APIs related to graphics:

- [Setting Graphics View Options](#)
- [Setting the Display Options for Annotations](#)
- [Result Vector Display Options](#)
- [Manipulating Graphics](#)
- [Exporting Graphics](#)
- [Exporting Result or Probe Animations](#)
- [Creating Section Planes](#)
- [Inserting Image Plane](#)
- [Setting Model Lighting Properties](#)
- [Manage Views with Model View Manager](#)

Setting Graphics View Options

You can use this command to access options for showing or hiding a few objects in the graphics view:

```
Graphics.ViewOptions
```

Supported graphical view properties are listed below.

Property	Description
ShowLegend	Gets or sets whether a legend is shown for the result object.
ShowTriad	Gets or sets whether the triad is shown.
ShowRuler	Gets or sets whether the ruler is shown.
Reset	Resets the graphics view properties to the defaults.
ShowMesh	Gets or sets whether to display the model's mesh.
ShowRandomColors	Gets or sets whether to show random colors for each object of the application.
ShowVertices	Gets or sets whether to display all the vertices of the model.
ShowClustered-Vertices	Gets or sets whether to display all closely clustered vertices of the model.
ShowEdgeDirection	Gets or sets whether to display the edge direction arrow.
ShowMeshConnection	Gets or sets whether to display edge connection using a color scheme based on the mesh connection information.
ShowThickEdgeScoping	Gets or sets whether to thicken the display of edge scoping.

Property	Description
ModelDisplay	Gets or sets the model display options.
ModelColoring	Gets or sets the model display coloring options.
RescaleAnnotations	Rescales the size of annotations following a zoom in or zoom out of the model.
ShowCoordinateSystems	Gets or sets whether to display all defined coordinate systems.
ClusteredVertexTolerance	Gets or set the clustered vertices tolerance value.
ShowBeamThickness	Gets or sets whether to display the thickness of beams.
ShowShellThickness	Gets or sets whether to display the thickness of shells.
ShowSPHExpansion	Gets or sets whether to display the expansion for sph elements.
ShowRemotePointConnections	Gets or sets whether to display the Remote Point Connections.

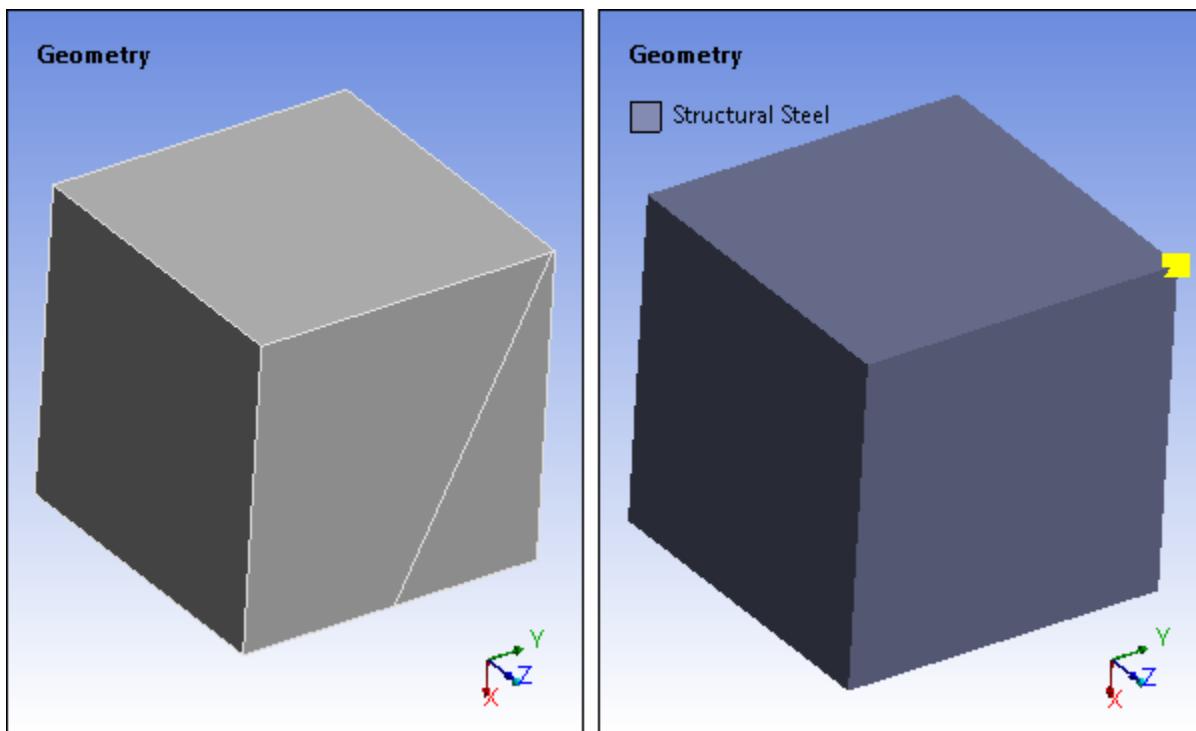
Code Examples

This code hides the ruler and the triad and shows the legend for the active result:

```
Graphics.ViewOptions.ShowRuler = False
Graphics.ViewOptions.ShowLegend = True
Graphics.ViewOptions.ShowTriad = False
```

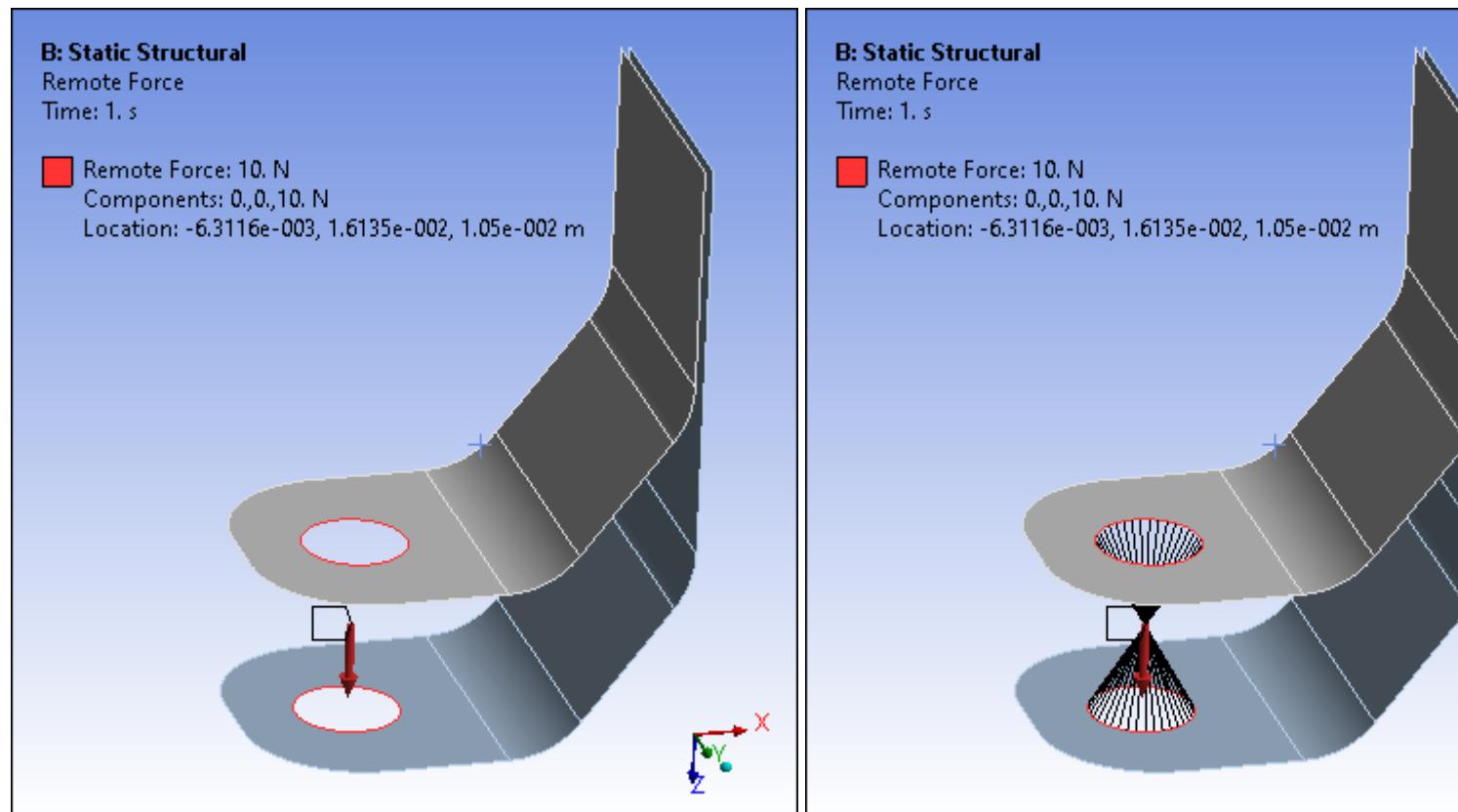
This code modifies the global view options settings by rescaling the annotations, setting the model display by shaded exterior, model coloring by material, setting the clustered vertex tolerance to 0.5 and displaying the clustered vertices:

```
Graphics.ViewOptions.RescaleAnnotations()
Graphics.ViewOptions.ModelDisplay = ModelDisplay.ShadedExterior
Graphics.ViewOptions.ModelColoring = ModelColoring.ByMaterial
Graphics.ViewOptions.ClusteredVertexTolerance = Quantity( 0.5, 'm')
Graphics.ViewOptions.ShowClusteredVertices = True
```



This code displays connection lines between geometry and Remote Points or between supported remote boundary conditions and Remote Points.

```
Graphics.ViewOptions.ShowRemotePointConnections = True
```



For information about legend settings for Mechanical results, see [Working with Legends \(p. 142\)](#).

Setting the Display Options for Annotations

Annotation Preference properties enable you to control the display of annotations in the **Geometry** window. You access the properties using the following command entry. The properties are described below as well as [examples \(p. 97\)](#) of the use of certain properties.

```
Graphics.ViewOptions.AnnotationPreferences
```

Available display properties include:

Property	The Property Gets or Sets the Visibility of...
ShowAllAnnotations	All annotations.
ShowCustomAnnotations	User-defined annotations.
ShowLabels	Annotation labels.
ShowPointMasses	Point Mass annotations.
ShowBeams	Beam annotations.
ShowSprings	Spring annotations.
ShowBearings	Bearing annotations.
ShowCracks	Crack annotations.
ShowForceArrows	Force arrows on surface reaction.
ShowBodyScopings	Body scoping annotations.
ShowMeshAnnotations	Mesh node and mesh element annotations in Named Selection displays.
ShowNodeNumbers	Mesh node numbers in Named Selection, Mesh, and Result displays.
ShowElementNumbers	Mesh element numbers in Named Selection, Mesh, and Result displays.
ShowNamedSelectionElements	Elements for all items in Named Selections group.

Available sizing properties include:

Property	Description
PointMassSize	Gets or sets the size for point mass annotation. (Small -Large; 1-100).
SpringSize	Gets or sets the size for spring annotation. (Small - Large; 1-100).

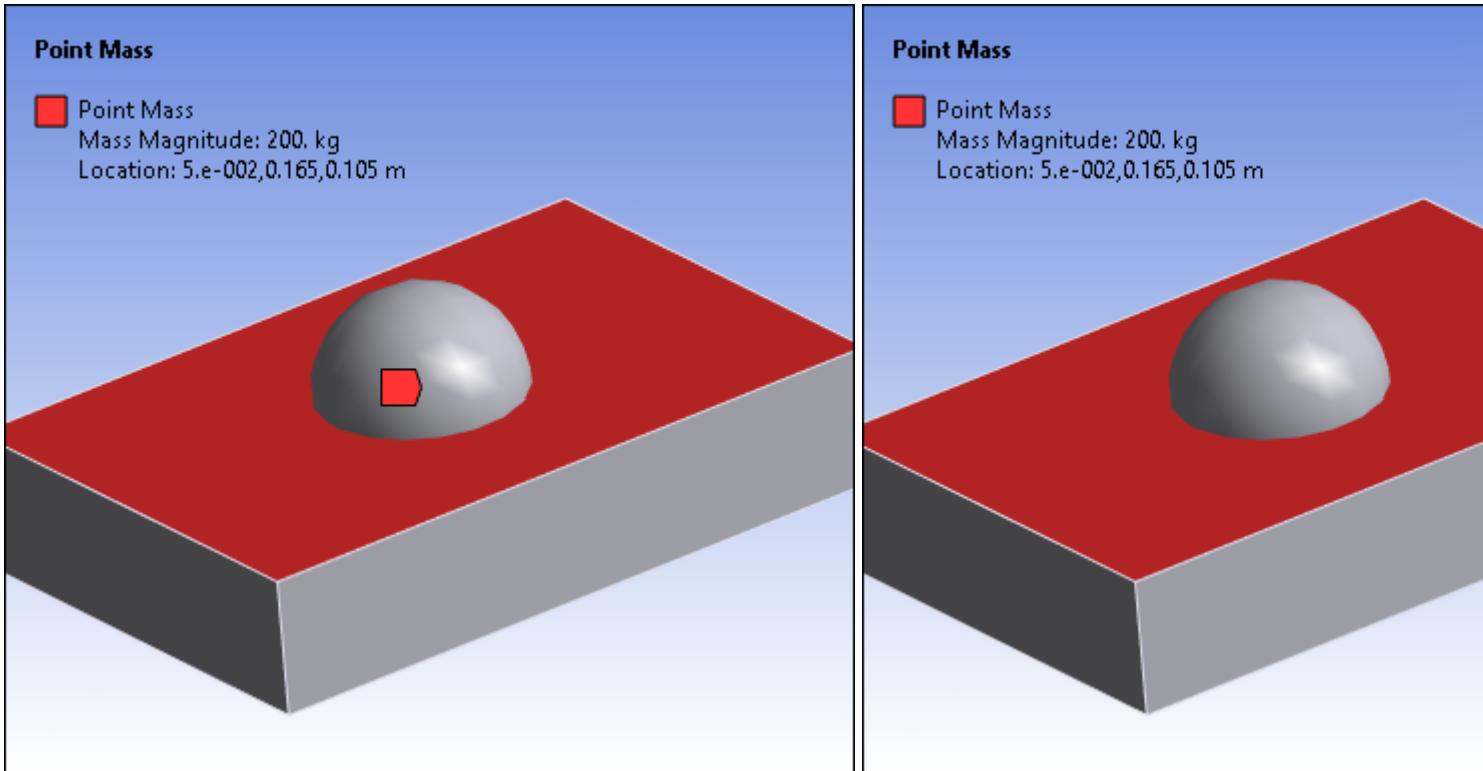
The following methods enables you to specify annotation node and element numbering.

Property	Description
SetNodeNumbering	Sets the begin, end, and increment values to display node numbering.
SetElementNumbering	Sets the begin, end, and increment values to display element numbering.

Code Examples

This code example sets the visibility of all annotation labels to false and the Point Mass annotation to true.

```
Graphics.ViewOptions.AnnotationPreferences.ShowLabels = False
Graphics.ViewOptions.AnnotationPreferences.ShowPointMasses = True
```

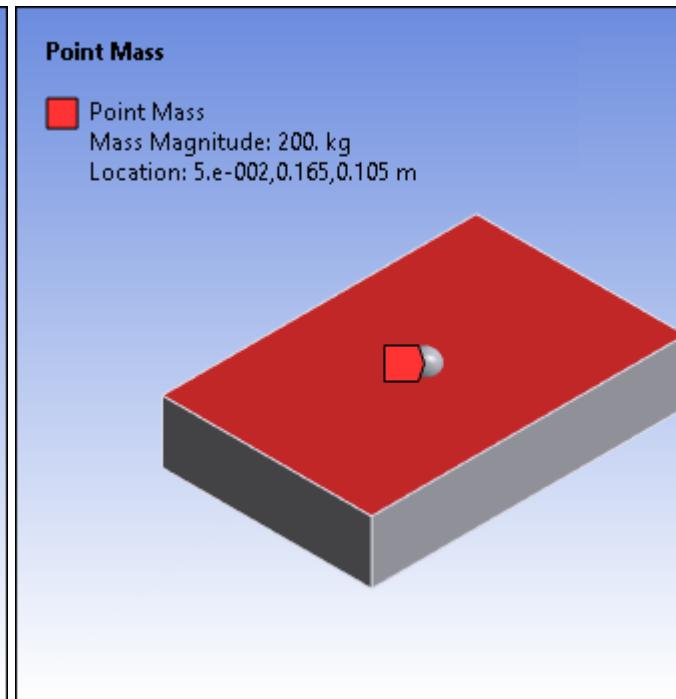
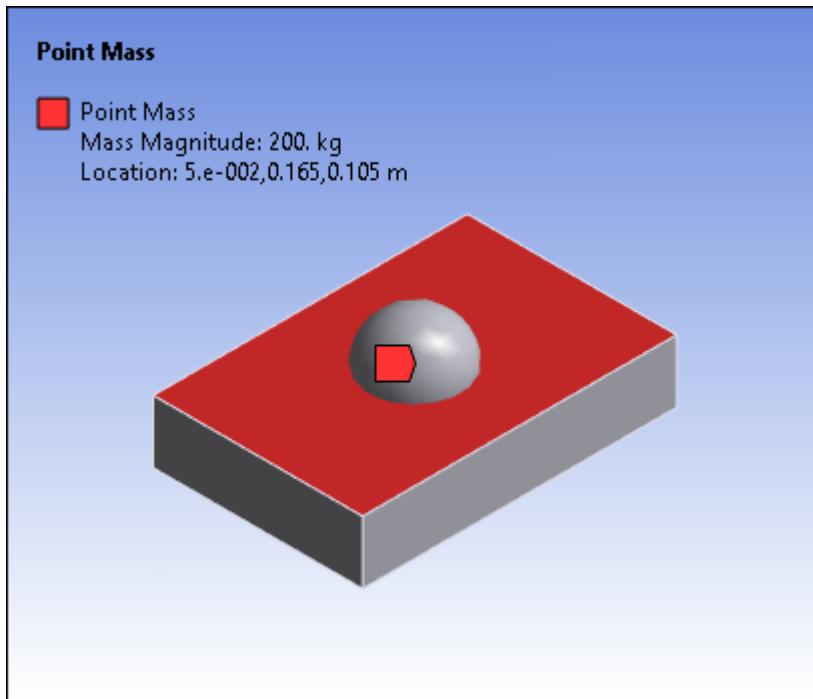


This code sets the size of Point Mass sphere. The supported size range is between 1-100. Although not entered as a value, this sizing is also available using the [Preferences](#) option in the [Annotation](#) group on the [Display](#) tab.

```
Graphics.ViewOptions.AnnotationPreferences.PointMassSize = 20
```

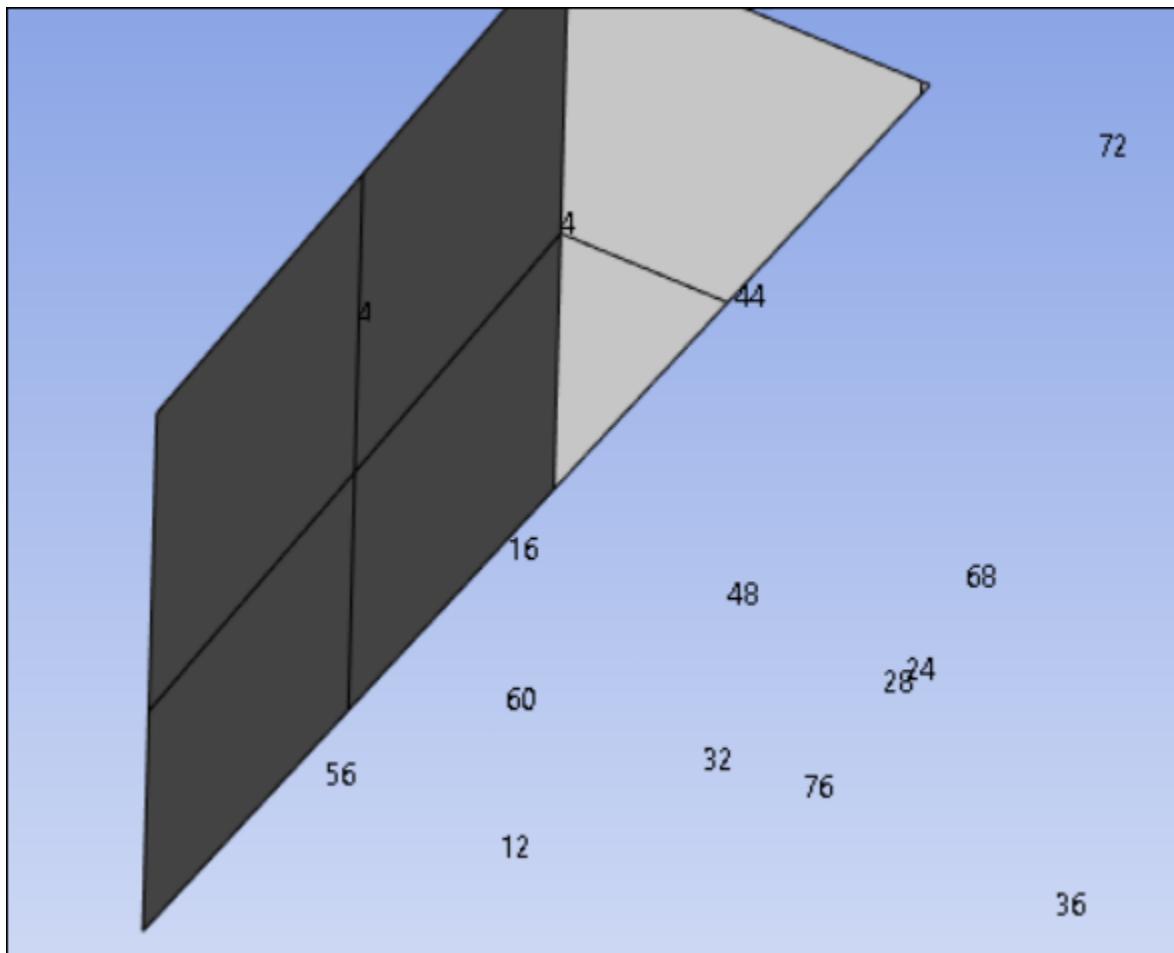
Point Mass Size = 100

Point Mass Size = 20



This code example sets the visibility to show node numbers to true and displays the first 76 node numbers incremented by a value 4.

```
Graphics.ViewOptions.AnnotationPreferences.ShowNodeNumbers = True  
Graphics.ViewOptions.AnnotationPreferences.SetNodeNumbering(1,76,4)
```



Result Vector Display Options

Under certain conditions, results can be displayed as vectors, and these vectors can be modified through result vector display options.

To enable or disable result vectors, you use the command `ShowResultVectors`:

```
Graphics.ViewOptions.ShowResultVectors
```

Additionally, to access result vector display options, you use the command `VectorDisplay`:

```
Graphics.ViewOptions.VectorDisplay
```

The following properties are available for the `VectorDisplay` object:

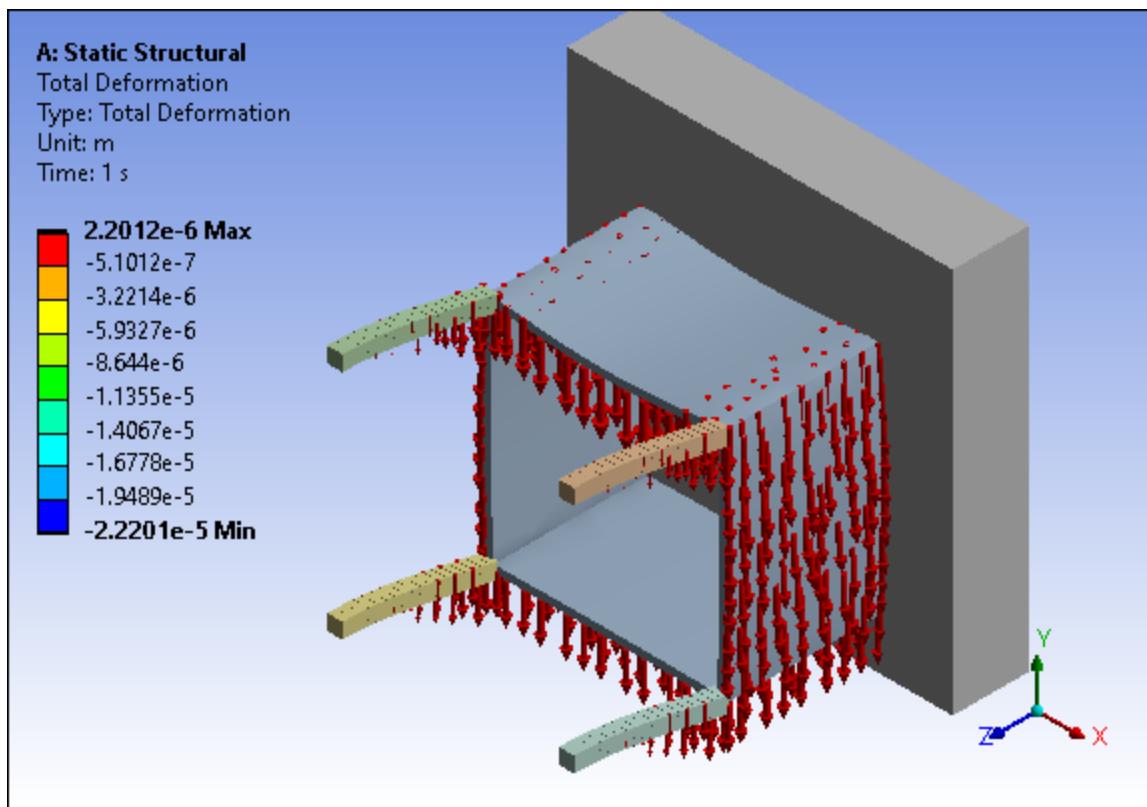
Property	Description
<code>LengthType</code>	Gets or sets the result vector length type.
<code>DisplayType</code>	Gets or sets the result vector display type.
<code>ControlDisplayDensity</code>	Gets or sets the ability to manually control the vector display density.
<code>DisplayDensity</code>	Gets or sets the vector display density in percentage.

Property	Description
LengthMultiplier	Gets or sets the vector length multiplier.
ShowTriadXAxis	Gets or sets whether to display the X axis vector of the triad/tensor.
ShowTriadYAxis	Gets or sets whether to display the Y axis vector of the triad/tensor.
ShowTriadZAxis	Gets or sets whether to display the Z axis vector of the triad/tensor.

Code Examples

This code modifies the result vectors by changing the length type to proportional, setting the display type to solid, setting the display density to 30.0%, and setting the length multiplier to 4.0:

```
Graphics.ViewOptions.VectorDisplay.LengthType = VectorLengthType.Proportional
Graphics.ViewOptions.VectorDisplay.DisplayType = VectorDisplayType.Solid
Graphics.ViewOptions.VectorDisplay.ControlDisplayDensity = True
Graphics.ViewOptions.VectorDisplay.DisplayDensity = 30.0
Graphics.ViewOptions.VectorDisplay.LengthMultiplier = 4.0
```



Manipulating Graphics

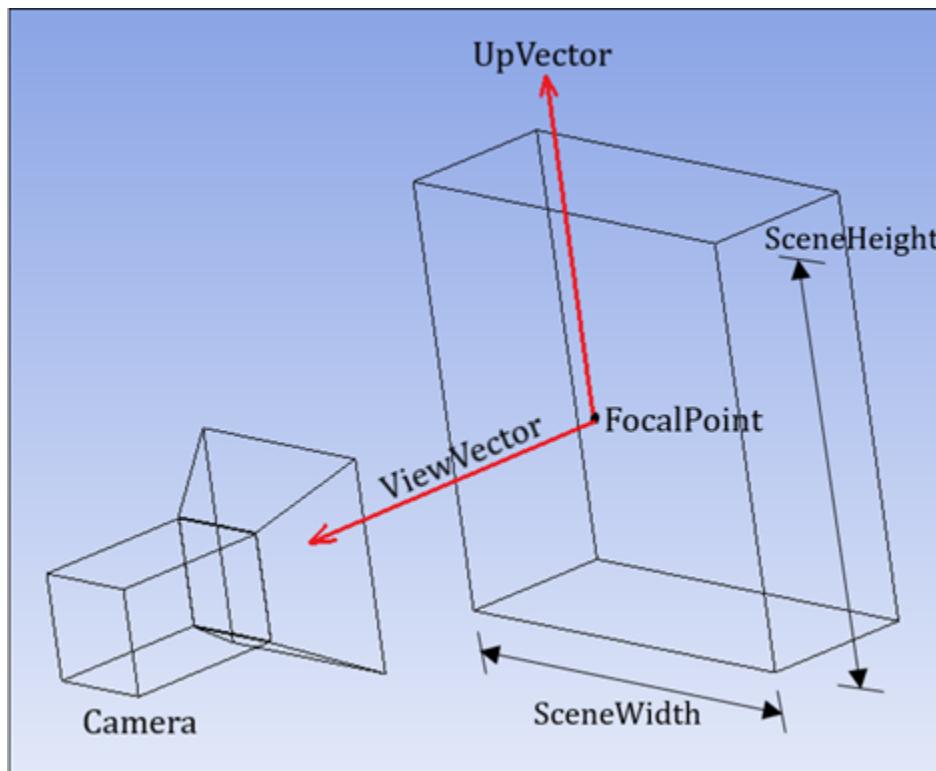
You can use the [MechanicalCameraWrapper](#) to manipulate the camera for precise control of model visualization.

To access this API, you enter these commands in the console:

```
camera = Graphics.Camera
camera
```

The following properties together represent the state of the camera view:

Property	Description
FocalPoint	Gets or sets the focal point of the camera. Coordinates are in the global coordinate system.
SceneHeight	Gets or sets the scene height (in length units) that will be projected and fit to the viewport.
SceneWidth	Gets or sets the scene width (in length units) that will be projected and fit to the viewport.
UpVector	Gets or sets the vector pointing up from the focal point.
ViewVector	Gets or sets the vector pointing from the focal point to the camera.



An example follows for using these properties to change the camera state:

```
camera.FocalPoint = Point((0.0,0.0,0.0,0.0), "mm")
camera.ViewVector = Vector3D(1.0,0.0,0.0)
camera.UpVector = Vector3D(0.0,1.0,0.0)
camera.SceneHeight = Quantity(100, "mm")
camera.SceneWidth = Quantity(150, "mm")
```

In addition to changing one or more specific properties, user-friendly methods are available for manipulating the camera state.

- To fit the view to the whole model or selection:

```
camera.SetFit(ISelectionInfo selection = null)
```

- To rotate along a particular axis (global or screen):

```
camera.Rotate(double angle, CameraAxisType axisType)
```

- To set a specific view orientation:

```
camera.SetSpecificViewOrientation(ViewOrientationType orientationType)
```

Two examples follow.

Example 1

This code rotates the model by 30 degrees about the direction normal to the current screen display:

```
camera.Rotate(30, CameraAxisType.ScreenZ)
```

Example 2

These code samples fit the view to a selection (either a face with a known ID or a named selection).

```
#fit view to face #28
selection = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
selection.Ids = [28]
camera.SetFit(selection)

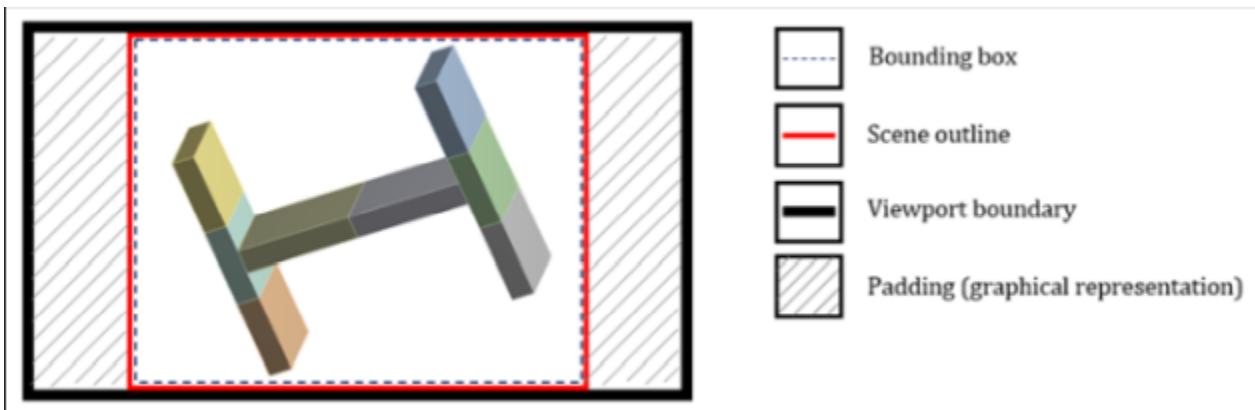
#fit view to the first named selection
named_selection = Model.NamedSelections.Children[0]
camera.SetFit(named_selection)
```

SceneHeight and SceneWidth Usage

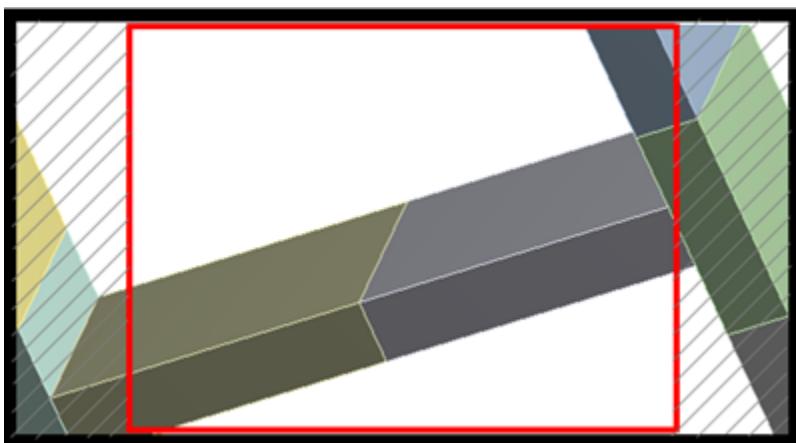
The two properties **SceneHeight** and **SceneWidth**, along with the existing camera APIs (such as **FocalPoint**, **UpVector**, and **ViewVector**) determine the volume of the scene. This volume defines the minimum view that will be visible (without any distortion) after projection onto a 2D viewport in graphics (which may have a different aspect ratio.) These quantities are in length units and can only be affected by zoom operations like zoom in/out, zoom to fit, and box zoom. This means that for a graphics window of a given aspect ratio, the extent defined by these two properties will always be seen on the screen (sometimes with additional “padding” depending on the aspect ratio).

An example scenario follows:

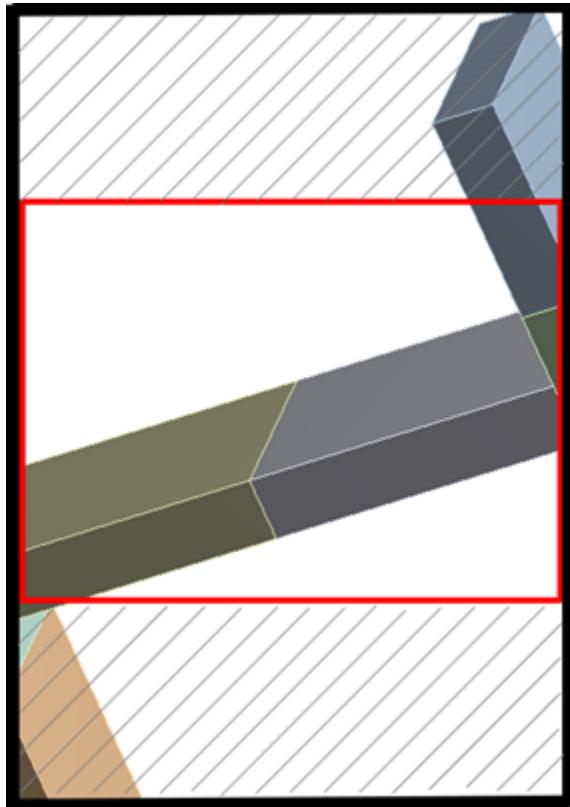
1. Open a model and zoom to fit. The **SceneHeight** and **SceneWidth** are equal to the bounding box of the directed view. There is horizontal padding because the viewport boundary is larger than the bounding box in that dimension.



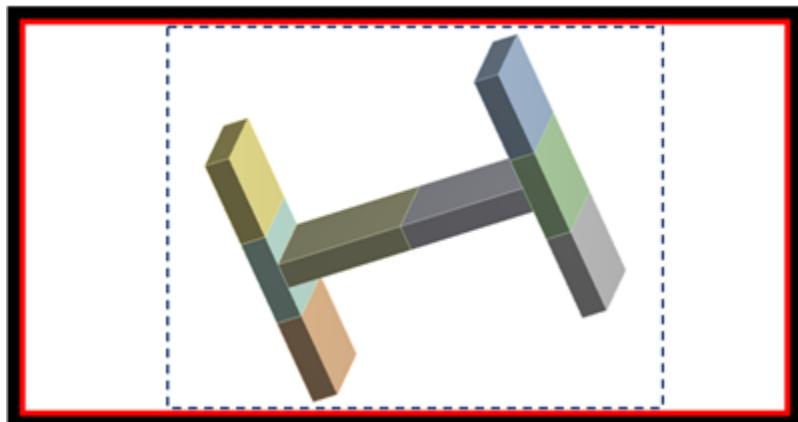
- If you set **SceneHeight** and **SceneWidth** to one-third () of their original values, the camera zooms in. The deciding factor is **SceneHeight** because of the screen's aspect ratio and orientation. The scene, defined by **SceneWidth** and **SceneHeight**, is then scaled in or out, maintaining the aspect ratio until the height equals the viewport height.



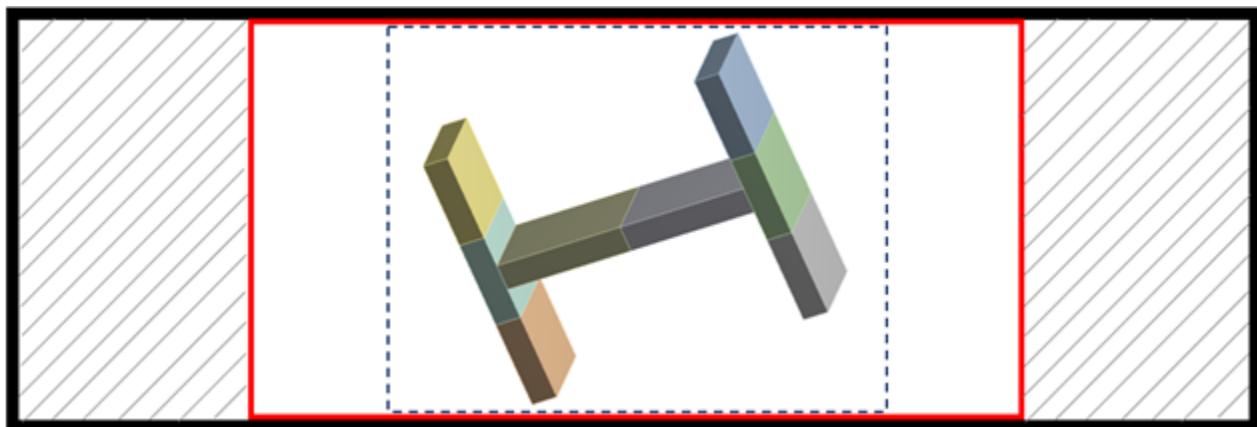
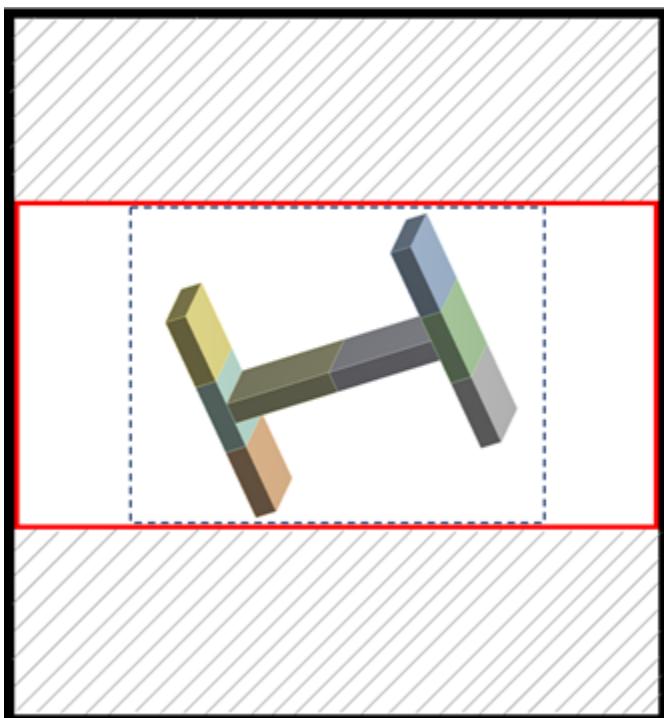
- If you now resize the viewport panel to be taller and more slender, the scene defined by **SceneHeight** and **SceneWidth** is then scaled in or out, maintaining the aspect ratio until the width equals the viewport width.



4. Let's say after point 1, you increase the **SceneWidth** to something beyond the model bounding box (such that it includes the padding space).

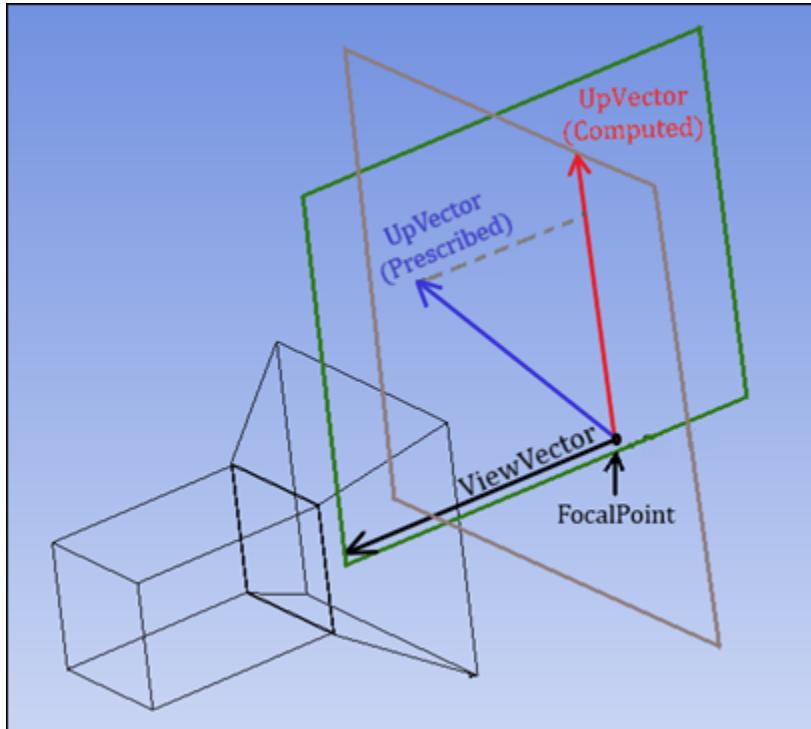


5. If you enlarge the viewport panel in either direction, the scene parameters still hold their shape.



UpVector Usage

Mathematically, the **UpVector** and **ViewVector** of the camera must be perpendicular. Together they define the camera orientation, which in the following figure is shown by the green half-plane. However, you can use the API to prescribe any **UpVector** that is not collinear with the **ViewVector**. The camera orientation's **UpVector** is then internally computed to use the projection (dashed line) of the prescribed **UpVector** (blue line) onto the brown plane perpendicular to the **ViewVector** (black line). The camera still reports back the prescribed **UpVector** with the **UpVector** property.

**Note:**

In the half-plane, there are an infinite number of prescribed **UpVector** choices that result in the same computed **UpVector**.

Exporting Graphics

You can use the methods **Export3D** and **ExportImage** to export graphics to 3D model files (STL and AVZ) and image files (PNG, JPG, TIF, BMP, and EPS).

To export graphics to model and image files, you use the following commands:

Command	Description
Graphics.Export3D	Exports the model to an STL or AVZ file.
Graphics.ExportImage ^[a]	Exports the image to a PNG, JPG, TIF, BMP, or EPS file.

[a] This method is supported for no GUI mode in Workbench.

This code exports the model to a binary STL file:

```
setting3d = Ansys.Mechanical.Graphics.Graphics3DExportSettings()
Graphics.Export3D("c:\\temp\\binarySTL.stl", Graphics3DExportFormat.BinarySTL, setting3d)
```

This code exports the image to a PNG file:

```
setting2d = Ansys.Mechanical.Graphics.GraphicsImageExportSettings()
Graphics.ExportImage("c:\\temp\\imagePNG.png", GraphicsImageExportFormat.PNG, setting2d)
```

For information on changing the 3D model or image export settings, see [Graphics3DExportSettings](#) or [GraphicsImageExportSettings](#) in the [ACT API Reference Guide](#).

The following script exports the model with a white background to an AVZ file and exports the image with enhanced resolution to a PNG file.

```
#export model to AVZ file with white background
setting3d = Ansys.Mechanical.Graphics.Graphics3DExportSettings()
setting3d.Background = GraphicsBackgroundType.White
Graphics.Export3D("c:\\avz_white.avz", Graphics3DExportFormat.AVZ, setting3d)

#export image with enhanced resolution to PNG file
setting2d = Ansys.Mechanical.Graphics.GraphicsImageExportSettings()
setting2d.Resolution = GraphicsResolutionType EnhancedResolution
Graphics.ExportImage("c:\\temp\\image_enhancement.png", GraphicsImageExportFormat.PNG, setting2d)
```

Exporting Result or Probe Animations

You can use the method **ExportAnimation** to export a result or probe animation to a video format (MP4, WMV, AVI, or GIF).

When exporting an animation to a video file, you can set the file name, format, and export settings.

- **GraphicsAnimationExportFormat** sets the file format to which to save the animation.
- **AnimationExportSettings** sets the width and height properties for the file. When this is not set, the resolution for the current graphics display is used to set the width and height.

Example 1

This code exports a result animation to a MP4 video file:

```
#Exporting a result animation to mp4 video file
totalDeform = DataModel.GetObjectsByName("Total Deformation")[0]
totalDeform.ExportAnimation("E:\\file.mp4",GraphicsAnimationExportFormat.MP4)
```

Example 2

This code exports a result animation to a WMV file with a specific resolution (width=1000,height=665):

```
#Exporting a result animation to wmv video file with given size
totalDeform1 = DataModel.GetObjectsByName("Total Deformation 1")[0]
settings = Ansys.Mechanical.Graphics.AnimationExportSettings(width = 1000, height = 665)
totalDeform1.ExportAnimation("E:\\file.wmv",GraphicsAnimationExportFormat.WMV,settings)
```

Animation Settings

To control the behavior of the animation for all results, you can specify global animation settings related to the toolbar visible when the **Animation** feature is used:

Property	Description
Graphics.ResultAnimationOptions.NumberOfFrames	Gets or sets the number of frames for the distributed result animation range type.

Property	Description
<code>Graphics.ResultAnimationOptions.Duration</code>	Gets or sets the range type of the result animation.
<code>Graphics.ResultAnimationOptions.TimeDecayCycles</code>	Gets or sets the number of cycles for the time decay analysis.
<code>Graphics.ResultAnimationOptions.UpdateContourRangeAtEachFrame</code>	Gets or sets if the legend contours update at each frame.
<code>Graphics.ResultAnimationOptions.FitDeformationScalingToAnimation</code>	Gets or sets the fit deformation scaling at each range for the full range for a result that has multiple time steps.

This code sets the number of frames to 200:

```
#set the number of frames to 200
Graphics.ResultAnimationOptions.NumberOfFrames = 200
```

This code sets the play duration to 12 seconds:

```
#set play duration to 12 seconds
Graphics.ResultAnimationOptions.Duration = Quantity(12, "s")
```

Creating Section Planes

You can use the **SectionPlanes** object, which is a collection of individual section planes, to add a section plane on your model to view the interior of the geometry, mesh, or results or to view the shape of the cross section. The **SectionPlanes** collection can have any number of section planes, but no more than six section planes can be active at once.

To access the collection associated with the **Section Planes window**, you use this command:

```
Graphics.SectionPlanes
```

The items in the collection use the **SectionPlane** object. The following methods and properties can be used to manipulate the collection:

Command	Description
<code>SectionPlanes.Add(SectionPlane)</code>	Add a new section plane to the collection.
<code>SectionPlanes.Remove(SectionPlane)</code>	Remove a new section plane from the collection.
<code>SectionPlanes.RemoveAt(0)</code>	Remove a section plane at a given index.
<code>SectionPlanes.Clear()</code>	Clear all the section planes from the collection.
<code>section_plane = SectionPlanes[0]</code>	Get a section plane at an index.

Along with methods and properties to manipulate the collection, the **SectionPlanes** object has two global properties that apply to all section planes. These global properties are shown in the **Graphics** window:

Command	Description
SectionPlanes.Capping	Capping style of the section plane.
SectionPlanes.ShowWholeElement	Element visibility of the section plane.

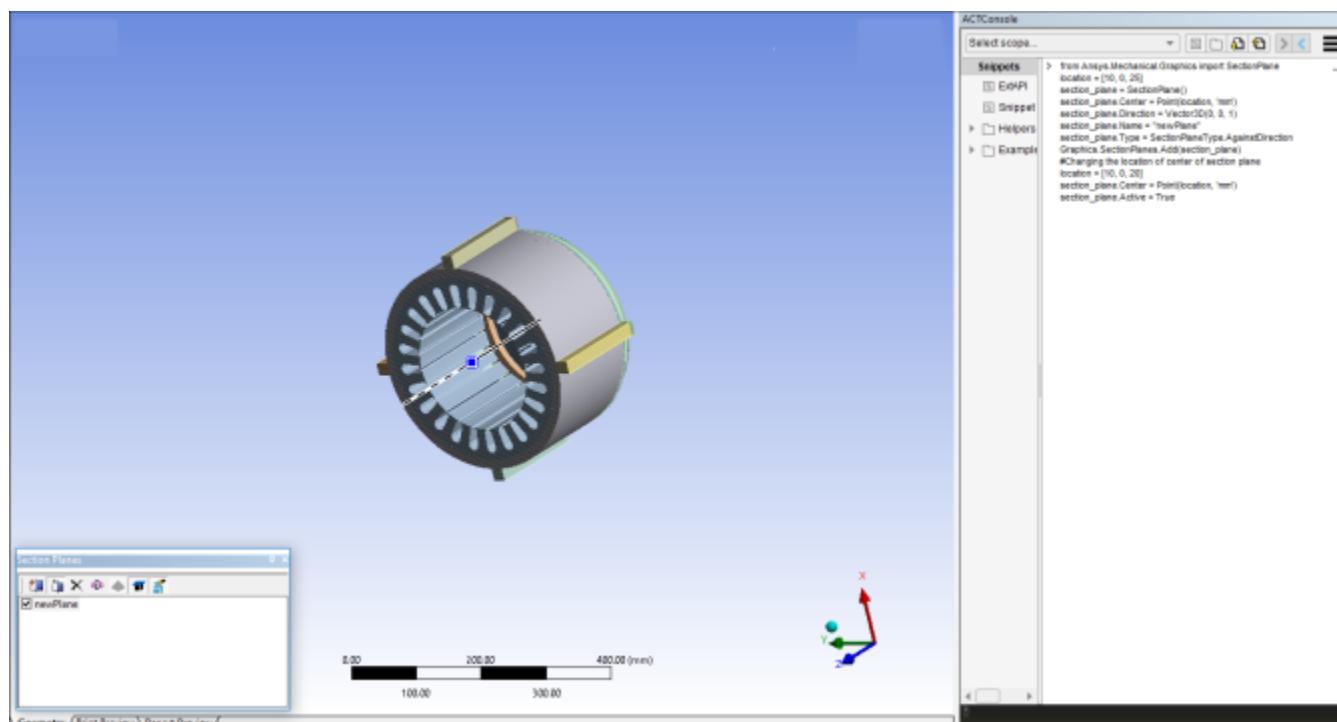
Each individual section plane in the collection contains these properties:

Command	Description
Graphics.SectionPlanes[0].Center	Center point of the section plane.
Graphics.SectionPlanes[0].Type	Type of the section plane.
Graphics.SectionPlanes[0].Name	Name of the section plane.
Graphics.SectionPlanes[0].Active	Active state of the section plane.
Graphics.SectionPlanes[0].Direction	Normal direction of the section plane

Example 1

This code creates a section plane and then both adds and changes the location:

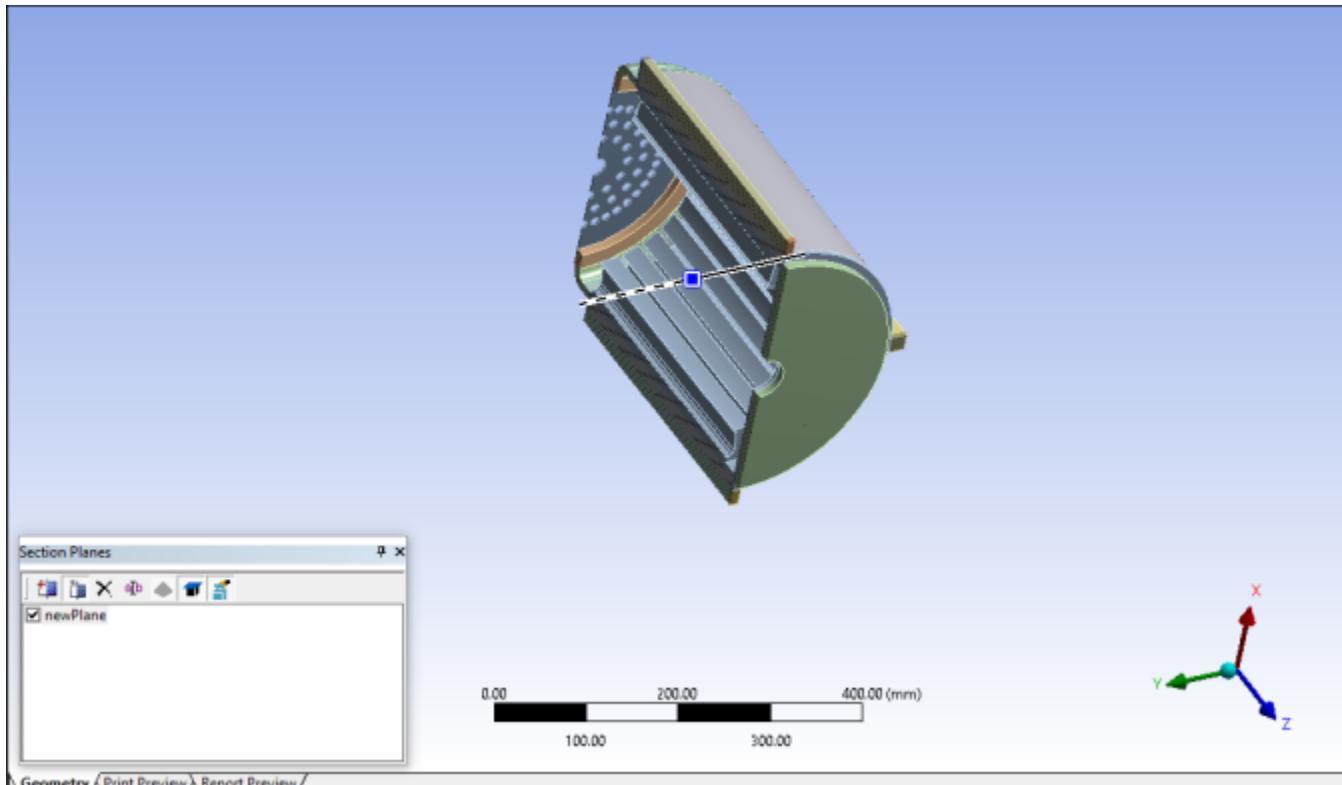
```
from Ansys.Mechanical.Graphics import SectionPlane
location = [100, 150, 255]
sectionPlane = SectionPlane()
sectionPlane.Center = Point(location, 'mm')
sectionPlane.Direction = Vector3D(0, 0, 1)
sectionPlane.Name = "newPlane"
sectionPlane.Type = SectionPlaneType.AgainstDirection
Graphics.SectionPlanes.Add(sectionPlane)
location = [100, 150, 200]
sectionPlane.Center = Point(location, 'mm')
```



Example 2

This code modifies the direction of the section plane:

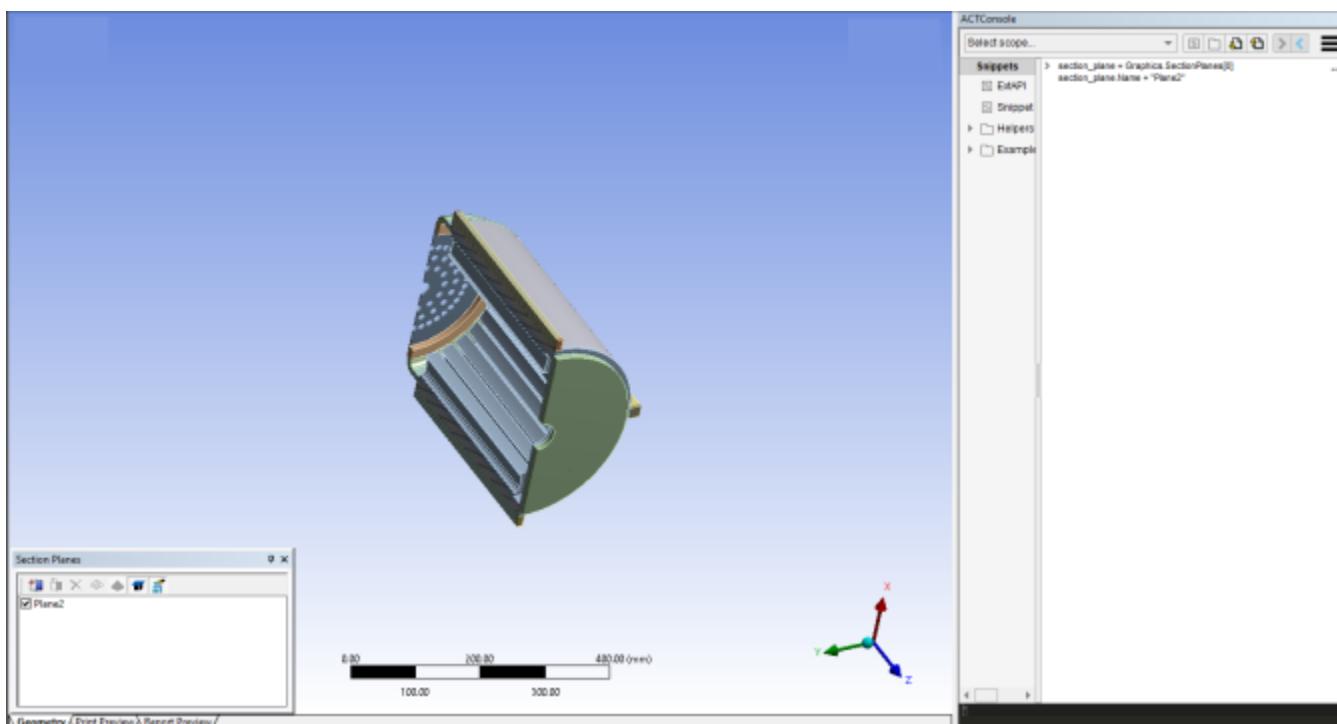
```
sectionPlane.Direction = Vector3D(0, 1, 0)
```



Example 3

This code gets the existing section plane and changes its name:

```
sectionPlane = Graphics.SectionPlanes[1]
sectionPlane.Name = "Plane2"
```



Example 4

As indicated earlier, while a **SectionPlanes** collection can have any number of **SectionPlane** objects, no more than six **SectionPlane** objects can be active at once. Assume that your collection holds six **SectionPlane** objects, which are all in an active state. If you try to add a new **SectionPlane** object with the property **Active** set to **True**, the **SectionPlane** object is added to the collection. However, the property **Active** is automatically set to **False** and an error message is shown:

```
> ExtAPI.Graphics.SectionPlanes.Count
< 6
> from Ansys.Mechanical.Graphics import SectionPlane
location = [100, 150, 255]
sectionPlane = SectionPlane()
sectionPlane.Center = Point(location, 'mm')
sectionPlane.Direction = Vector3D(0, 0, 1)
sectionPlane.Name = "newPlane"
sectionPlane.Type = SectionPlaneType.AgainstDirection
sectionPlane.Active = True
Graphics.SectionPlanes.Add(sectionPlane)
✖ Activation of the clip plane is not successful as the limit of 6 active clip planes has been reached.
> ExtAPI.Graphics.SectionPlanes.Count
< 7
> sectionPlane.Active
< False
```

While the collection now holds seven **SectionPlane** objects, only 6 are active. If you tried to activate the seventh **SectionPlane** object by setting its property **Active** to **True**, the property is automatically set back to **False** and an error message again displays that the limit of 6 active planes has been reached:

```
> ExtAPI.Graphics.SectionPlanes[6].Active
< False
> ExtAPI.Graphics.SectionPlanes[6].Active=True
✖ Activation of the clip plane is not successful as the limit of 6 active clip planes has been reached.
> ExtAPI.Graphics.SectionPlanes[6].Active
< False
```

Inserting Image Plane

You use the following command to insert an image object into the Outline. The object includes properties that enable you to specify the image's location in the Geometry window. You can place it on or around your model.

```
image_plane = model.AddImagePlane()
```

Properties	Description
Height	Gets or sets the physical height of the image plane.
Width	Gets or sets the physical width of the image plane.
PixelHeight	Gets the pixel height of the plane's image.
PixelWidth	Gets the pixel width of the plane's image.
ImageFile-Path	Gets the file path for the rendered image.
Import	Sets the image which is rendered to this plane, specified by filePath. Upon success, the ImageFilePath will be set to the filePath. Upon failure, ImageFilePath will remain unchanged. Supported file formats include PNG, JPEG, BMP, and GIF. If this is the first time an image has been set to this image plane and if Width and Height were never modified, the image plane will be automatically sized based on the model's bounding box, respecting the image's aspect ratio.
Translucency	Gets or sets the translucency of the plane, as a percentage ranging from 0 to 100.
CoordinateSystem	Gets or sets the coordinate system object used for defining the plane's location and orientation. This property is restricted to cartesian coordinate systems.
FlipHorizontally	Gets or sets whether the rendered image on the plane should be flipped horizontally.
FlipVertically	Gets or sets whether the rendered image on the plane should be flipped vertically.
ShowAlways	Gets or sets whether the image plane will be shown independent of the active object in the tree.

Properties	Description
CoordinateSystemVisible	Gets or sets whether the annotation for the image's coordinate system is rendered.

Example 1

The following code adds an image to the model.

```
# model is a reference to an instance of Ansys.ACT.Automation.Mechanical.Model
imageOne = model.AddImagePlane() # creates image plane object
imageOne.Import("C:\\example.png") # on success, sets ImageFilePath and renders the image on the plane
# image plane from screen shot of graphics window
imageTwo = model.AddImagePlane()
Graphics.ExportScreenToImage("C:\\ANSYSDev\\example2.png")
imageTwo.Import("C:\\ANSYSDev\\example2.png")
```

Example 2

The following code changes the location of the image in the Geometry window.

```
# modifying location and dimensional properties of the image plane
imageOne.CoordinateSystem = csysObj # sets the bottom left corner of the plane at the CSYS's location
imageOne.Width = Quantity(2.5, 'm') # sets world width of the plane
imageOne.Height = Quantity(250, 'cm') # sets world height of the plane
```

Example 3

The following code changes certain view properties of the object.

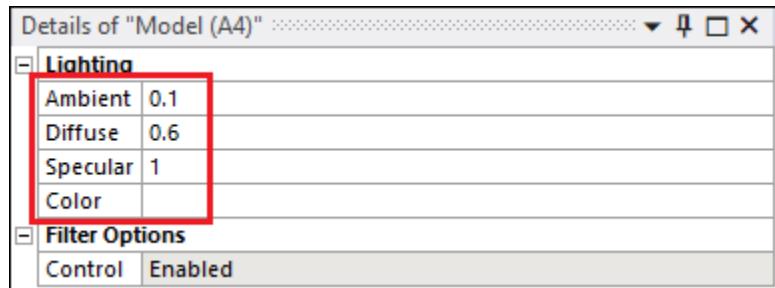
```
# modifying view properties of the image plane
imageOne.Translucency = 30 # sets the plane to 30% translucency (70% opacity)
imageOne.FlipHorizontally = True # flips the horizontal orientation of the rendered image (not of the physical plane)
imageOne.FlipVertically = False # flips the vertical orientation of the rendered image (not of the physical plane)
imageOne.CoordinateSystemVisible = True # annotates the coordinate system the image is attached to
ImageOne.ShowAlways = True # allows the image to be rendered no matter what object is active in the tree
```

Setting Model Lighting Properties

You use the following properties to set model lighting:

Property	Description
Ambient	Gets or sets the ambient lighting factor.
Diffuse	Gets or sets the diffuse lighting factor.
Specular	Gets or sets the specular lighting factor.
Color	Gets or sets the lighting color.

The following image shows model lighting properties being set in the Details view for the model:



Example 1

This code sets model lighting properties:

```
Model.Ambient = 0.5
Model.Diffuse = 0.3
Model.Specular = 0.2
Model.Color = 13796830
```

Example 2

When setting model lighting color, you can also use hex color codes, which are three-byte hexadecimal numbers consisting of the prefix **0x** followed by six digits. Each byte (or pair of characters) following the prefix represents the intensity of red, green, and blue, respectively. For example, the hex color code for white is **0xFFFFFFFF**.

This code shows how to use a hex color code to specify a lilac shade (■) for the model lighting color:

```
Model.Color = 0xD285DE
```

Manage Views with Model View Manager

You can programmatically control the **Model View** capability using the **ModelViewManager** data object which can be accessed via the **Graphics** Object as shown below.

```
view_manager = Graphics.ModelViewManager
```

To get the number of graphical views that are currently defined:

```
views_count = view_manager.NumberOfViews
```

You can create a graphical view from current graphics using either a default name or a specified name. To create a view using a default name:

```
view_manager.CreateView
```

The name assigned to the new view is **View** followed by the next sequential number. To create a view using a specified name, where **viewName** is the name to assign to the new view:

```
view_manager.CreateView(string viewName)
```

You can apply a graphical view by specifying its name or index. To apply a view by specifying its name, where `viewLabel` is the name of the view to apply:

```
view_manager.ApplyModelView(string viewLabel)
```

For example, to apply a view named Fit View:

```
view_manager.ApplyModelView("Fit View")
```

To apply a view by specifying its index, where `viewIndex` is the index of the listed view to apply:

```
view_manager.ApplyModelView(int viewIndex)
```

For example, to apply a view with index 1:

```
view_manager.ApplyModelView(1)
```

Additionally there are API's to cover the other features of Model View Manager such as:

Deleting a view:

```
DeleteView("Fit View")
```

Renaming a view:

```
RenameView("Fit View", "New Name")
```

Exporting a saved view list:

```
ExportModelViews("D:\My_Projects\Views\myviews.xml")
```

Importing a saved view list:

```
ImportModelViews("D:\My_Projects\Views\allviews.xml")
```


Results

In Mechanical, the **Solution** object contains results. To access the **Solution** object:

```
solution = Model.Analyses[0].Solution
```

The following topics describe ways of using the API to add and work with results:

[Adding Results to a Solution Object](#)

[Accessing Contour Results for an Evaluated Result](#)

[Set Result Display Options](#)

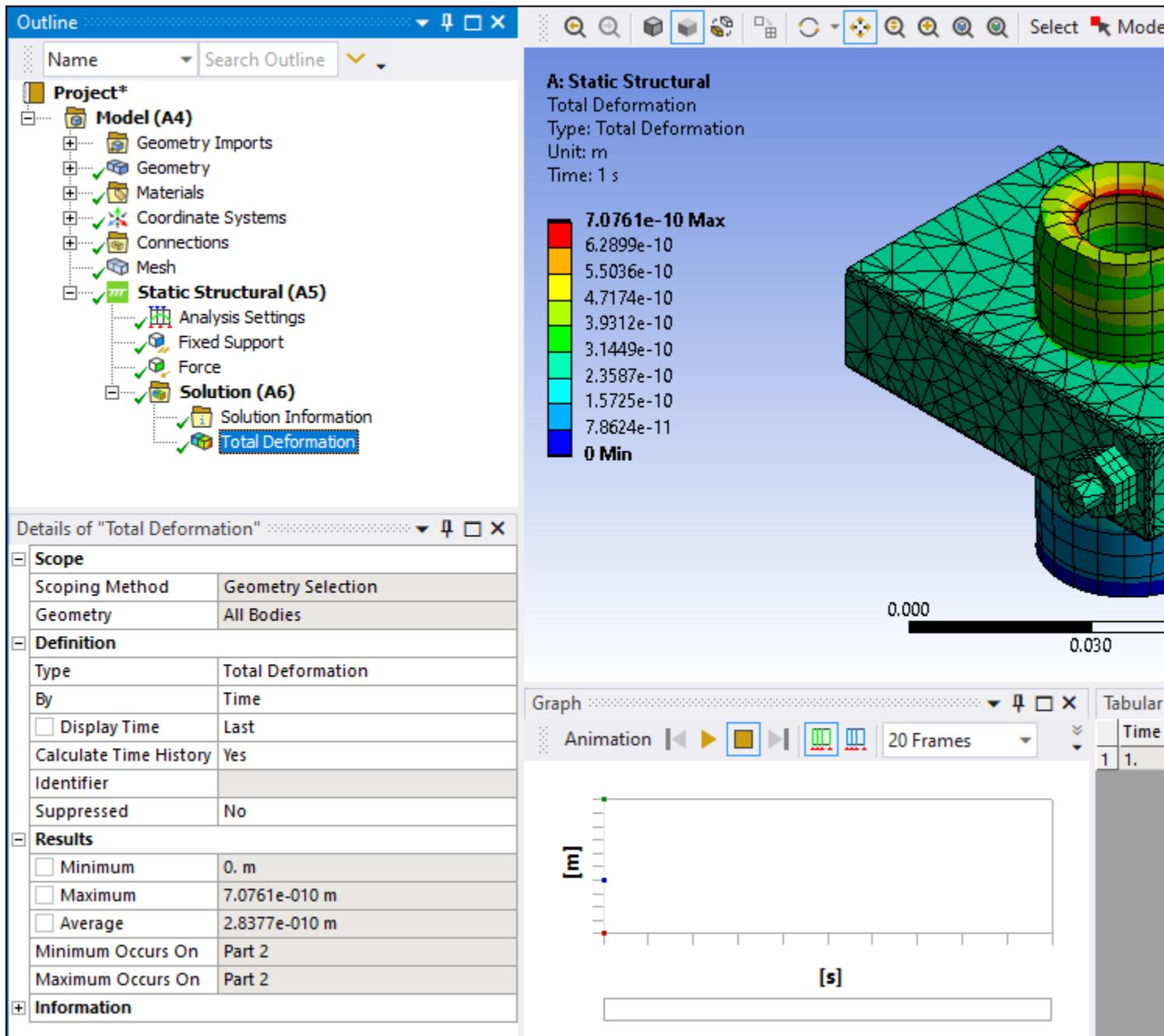
[Working with Legends](#)

Adding Results to a Solution Object

You can use the API to add results to the **Solution** object. For example, you can add the **Total Deformation** result to a static structural analysis and then retrieve the minimum and maximum total deformation:

```
total_deformation = solution.AddTotalDeformation()
analysis = Model.Analyses[0]
analysis.Solve(True)
minimum_deformation = total_deformation.Minimum
maximum_deformation = total_deformation.Maximum
```

It results in a solved analysis indicating the values for the properties **Minimum** and **Maximum** for the result **Total Deformation**.



Accessing Contour Results for an Evaluated Result

You can access contour results for an evaluated result in the tabular data interface. The result is represented in a table with independent and dependent column variables.

- The node and element IDs are independent variables (unique identifiers) of the tabular result.
- The result value components are dependent variables (based on the independent variables).

Contour results can be of three types:

- Nodal:** Results are calculated on each node (like displacements)

- **Elemental:** Results are calculated on each element (like volumes)
- **ElementalNodal:** Results are calculated on the element but then interpolated onto the nodes (like stresses)

Contour results can have various styles:

- **Scalar:** Has single result component
- **Vector:** Has X, Y, and Z components
- **Tensor:** Has X, Y, Z, XY, YZ, and XZ components
- **TensorStrain:** Has X, Y, Z, XY, YZ, and XZ components
- **EulerAngels:** Has XY, YZ, and XZ components
- **Coordinate:** Has X, Y, and Z components
- **ShearMomentDiagram:** Has MY,MZ,SFY,SFZ,UY,UZ, MSUM, SFSUM, and USUM components

See the following sections for more information:

[Accessing Contour Results from an Evaluated Result](#)

[Accessing Contour Results Scoped to Faces, Elements, or Nodes](#)

[Accessing Contour Results Scoped to Paths](#)

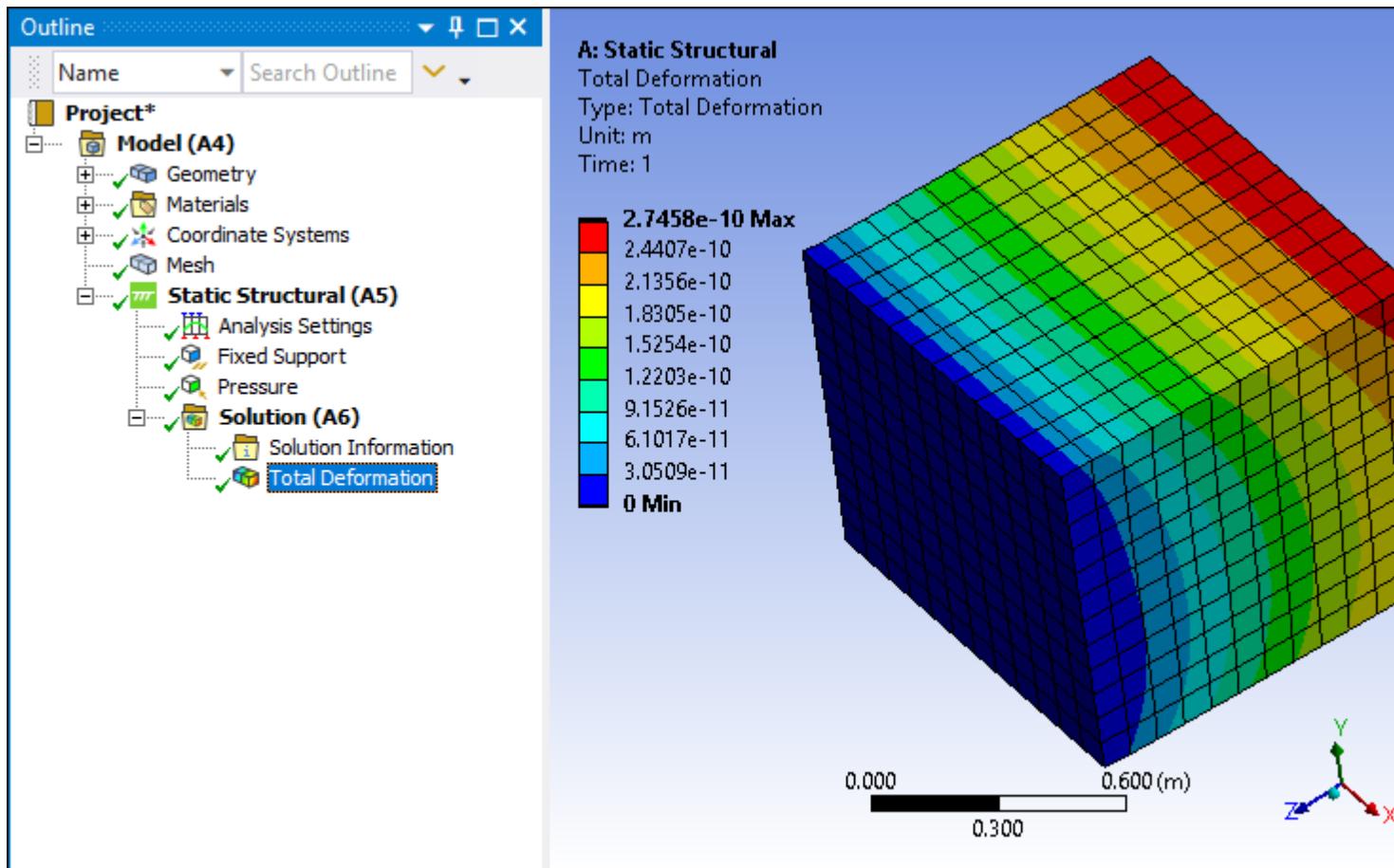
[Accessing Contour Results for Surfaces](#)

[Accessing Contour Results for Beams](#)

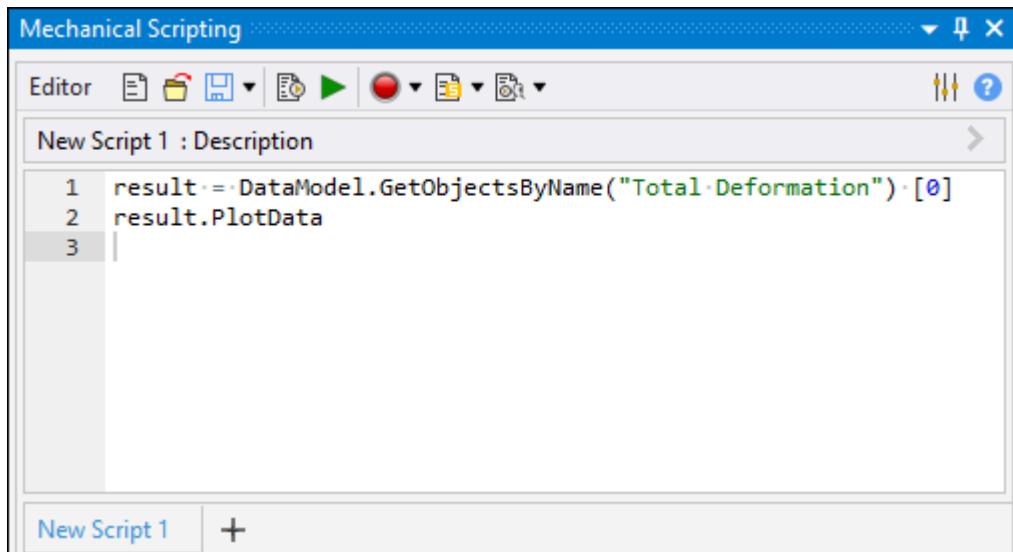
[Limitations of Tabular Data Interface](#)

Accessing Contour Results from an Evaluated Result

In Mechanical, you can create a contour result of a specific type and style and evaluate it. You can then access contour results from these evaluated results. For the Total Deformation result illustrated here, you use the **Mechanical Scripting** pane to display contour results in tabular format, as shown in the describes and examples below.



Using PlotData you return a result data table for a contour result.



For a **Nodal** result, the results table displays only node information.

The screenshot shows the Mechanical Scripting interface. The top window is titled "Mechanical Scripting" and contains a script editor with the following code:

```
1 result := DataModel.GetObjectsByName("Total Deformation")-[0]
2 result.PlotData
3 resultTable := result.PlotData
4 print(resultTable)
5
6
```

Below the script editor is a tab labeled "New Script 1" with a "+" button. The bottom window is titled "Shell" and displays the output of the executed script:

Script Executed		
	Node	Values
		m
0	1	2.4744E-10
1	2	2.472E-10
2	3	2.4707E-10
...		
8278	8279	2.3952E-10
8279	8280	2.4822E-10
8280	8281	2.5816E-10

For a **Elemental** result, the results table displays only node information.

Results

The screenshot shows the Mechanical Scripting interface. The top window is titled "Mechanical Scripting" and contains a script editor with the following code:

```
1 result := DataModel.GetObjectsByName("ENERGYPOTENTIAL")[0]
2 resultTable := result.PlotData
3 print(resultTable)
```

The bottom window is titled "Shell" and displays the output of the executed script:

Script Executed		
	Element	Values
		0
0	1	0.005304
1	2	0.0047967
2	3	0.0047312
...		
1725	1726	0.00052416
1726	1727	0.000525
1727	1728	0.00052299
>>>		

For a **Elemental Nodal** result, the result table has both node and element information (unaveraged results).

The screenshot shows the Mechanical Scripting interface. The top window is titled "Mechanical Scripting" and contains a script editor with the following code:

```
1 result := DataModel.GetObjectsByName("EPTTY")[0]
2 resultTable := result.PlotData
3 print(resultTable)
```

Below the script editor is a "Shell" window titled "Script Executed". It displays a table of contour results:

	Node	Element	Values
			m/m
0	1	1	6.5396E-06
1	2	2	5.8772E-06
2	3	3	5.8956E-06
...			
34557	8269	122	-6.0097E-08
34558	8270	121	-2.0564E-08
34559	8271	133	-4.702E-08

The shell window also shows a command prompt at the bottom: >>> |

Similarly, you can generate result table for contour results with multiple components.

The screenshot shows the Mechanical Scripting interface. The top window is titled "Mechanical Scripting" and contains a script editor with the following code:

```
1 result := DataModel.GetObjectsByName("UVECTORS")[0]
2 resultTable := result.PlotData
3 print(resultTable)
```

Below the script editor is a tab labeled "New Script 1" followed by a "+" button. The bottom window is titled "Shell" and displays the output of the executed script:

	Node	X	Y	Z
		m	m	m
0	1	-2.5366E-07	-4.9245E-07	6.2144E-07
1	2	-1.5337E-07	-4.653E-07	6.4745E-07
2	3	-9.949E-08	-4.6287E-07	6.5879E-07
...				
8278	8279	-5.3118E-07	-2.1396E-06	2.5217E-06
8279	8280	-5.7438E-07	-1.7703E-06	1.7578E-06
8280	8281	-5.5333E-07	-1.3131E-06	1.0474E-06

The shell window also shows a command prompt "">>>> |" at the bottom.

Example 1

The following script prints the independent columns of the result table.

The following script snippet prints the dependent columns of the result table.

The image shows two side-by-side "Mechanical Scripting" windows. Both windows have an "Editor" tab at the top with various icons. Below the editor is a "Description" pane.

Left Window (Independent):

```
1 result = DataModel.GetObjectsByName("UVECTORS")[0]
2 resultTable = result.PlotData
3 print(resultTable.Independents)
```

Right Window (Dependent):

```
1 result = DataModel.GetObjectsByName("UVECTORS")[0]
2 resultTable = result.PlotData
3 print(resultTable.Dependents)
4
```

Below the descriptions are "Shell" panes:

Left Shell:

Node	Script Executed
0	1
1	2
2	3
...	
8278	8279
8279	8280
8280	8281
>>>	

Right Shell:

	X	Y	Z
	m	m	m
0	-2.5366E-07	-4.9245E-07	6.2144E-07
1	-1.5337E-07	-4.653E-07	6.4745E-07
2	-9.949E-08	-4.6287E-07	6.5879E-07
...			
8278	-5.3118E-07	-2.1396E-06	2.5217E-06
8279	-5.7438E-07	-1.7703E-06	1.7578E-06
8280	-5.5333E-07	-1.3131E-06	1.0474E-06
>>>			

Example 2

The following script snippet prints the X component column, unit, count and a specific value of result table.

The screenshot shows the Mechanical Scripting interface. The top bar has tabs for 'Editor' and 'Shell'. The 'Editor' tab is active, displaying a script titled 'New Script 1 : Description' with the following code:

```
1 xComp = resultTable["X"]
2 print(xComp)
3 print(xComp.Unit)
4 print(xComp.Count)
5 print(xComp[5])
```

The 'Shell' tab is also active, showing the output of the script execution:

```
Script Executed
0 -2.5366E-07
1 -1.5337E-07
2 -9.949E-08
...
8278 -5.3118E-07
8279 -5.7438E-07
8280 -5.5333E-07
Count: 8281, Unit: "m"
m
8281
-1.01308061939e-09
>>> |
```

Example 3

The following script snippet prints the hidden column (body ids) along with the result table for a result.

The screenshot shows the Mechanical Scripting interface. The top window is titled "Mechanical Scripting" and contains a script editor with the following code:

```

1 resultTable.ShowHiddenColumns := True
2 print(resultTable)
3

```

The bottom window is titled "Shell" and displays the output of the executed script, which is a table of contour results:

	Body	Node	X m	Y m	Z m
0	16	1	-2.5366E-07	-4.9245E-07	6.2144E-07
1	16	2	-1.5337E-07	-4.653E-07	6.4745E-07
2	16	3	-9.949E-08	-4.6287E-07	6.5879E-07
...					
8278	16	8279	-5.3118E-07	-2.1396E-06	2.5217E-06
8279	16	8280	-5.7438E-07	-1.7703E-06	1.7578E-06
8280	16	8281	-5.5333E-07	-1.3131E-06	1.0474E-06
>>>					

Example 4

For the PlotData result shown here:

Node	X m	Y m	Z m
1	-2.5366E-07	-4.9245E-07	6.2144E-07
2	-1.5337E-07	-4.653E-07	6.4745E-07
3	-9.949E-08	-4.6287E-07	6.5879E-07
8279	-5.3118E-07	-2.1396E-06	2.5217E-06
8280	-5.7438E-07	-1.7703E-06	1.7578E-06
8281	-5.5333E-07	-1.3131E-06	1.0474E-06

The following script prints the third row of the result table for X and Y values by indexing each column.

The screenshot shows the Mechanical Scripting interface. At the top, there's a toolbar with icons for file operations like Open, Save, Print, and Run. Below the toolbar is a tab bar with "Independent : Description". The main area contains a code editor with the following Python script:

```
1 result = DataModel.GetObjectsByName("UVECTORS")[0]
2 resultTable = result.PlotData
3
4 node = resultTable["Node"][2]
5 xVal = resultTable["X"][2]
6 yVal = resultTable["Y"][2]
7
8 print("The x and y component of node {n} are : {x} {y}".format(n=node,x=xVal,y=yVal))
9
```

Below the code editor is a tab bar with "Independent" selected and a "+" button. The bottom part of the interface is a "Shell" window. It shows the output of the script execution:

```
Script Executed
The x and y component of node 3 are : -9.9e-08 -4.6e-07
>>> |
```

Example 5

The following script prints the Time Steps included in the Static Structural analysis.

The screenshot shows the Mechanical Scripting interface. At the top is a toolbar with icons for Editor, Save, Open, Print, Run, Stop, and Help. Below the toolbar is a script editor window titled "New Script 1 : Description". The code in the editor is:

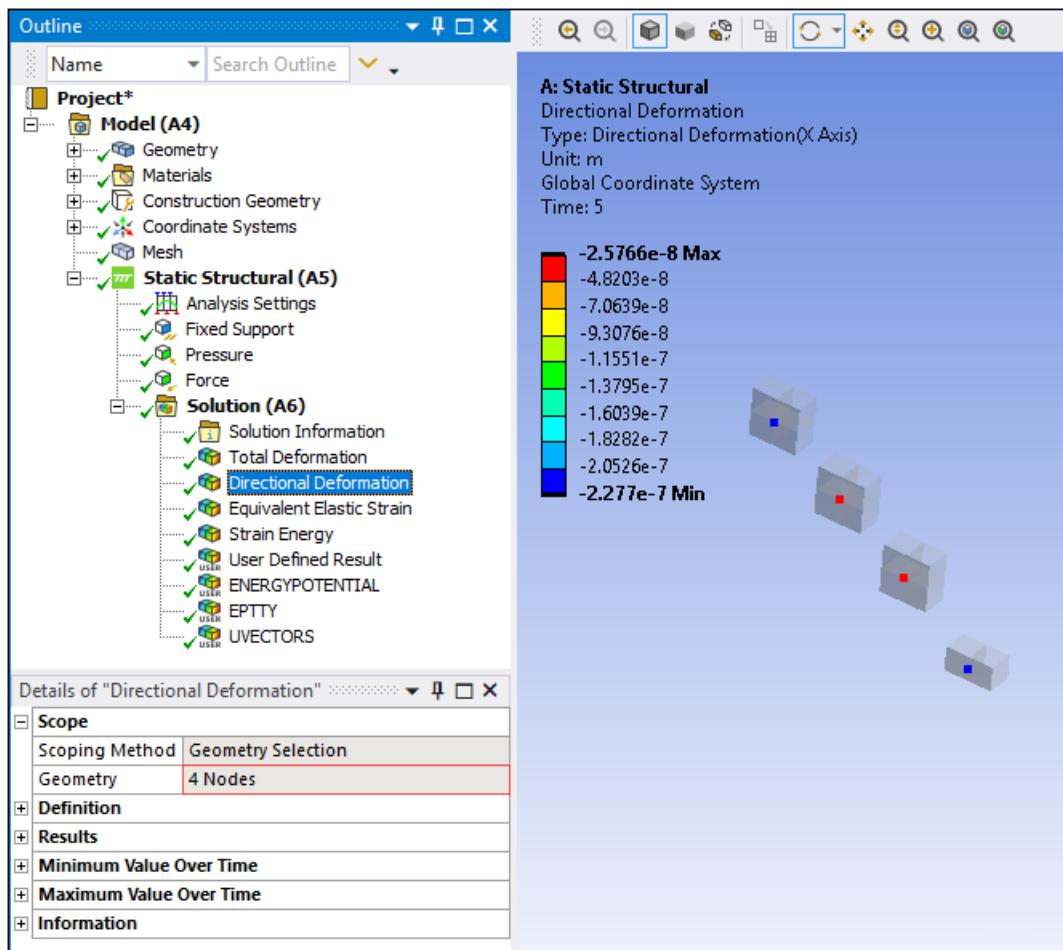
```
1 staticStructural := DataModel.GetObjectsByName("Static Structural")[0]
2 reader := staticStructural.GetResultsData()
3 print(reader.ListTimeFreq)
4
5
```

Below the editor is a shell window titled "Shell" with a "+" button. It displays the output of the script execution:

```
Script Executed
List[float]([1.0, 2.0, 3.0, 4.0, 5.0])
>>> |
```

Accessing Contour Results Scoped to Faces, Elements, or Nodes

You can scope contour results to an element face or a geometry face, or if they can be scoped to elements or nodes:



After the contour result scoped to specific entity is evaluated, the following code displays them in tabular format.

The screenshot shows the Mechanical Scripting interface. At the top is a toolbar with icons for file operations (New, Open, Save, Print, Run, Stop, Refresh) and other tools. Below the toolbar is a script editor titled "New Script 1 : Description" containing the following Python-like code:

```
1 result = DataModel.GetObjectsByName("Directional Deformation")[0]
2 resultTable = result.PlotData
3 print(resultTable)
```

Below the script editor is a tab labeled "New Script 1" with a "+" button to add more scripts. The main workspace is a "Shell" window titled "Script Executed". It displays the output of the executed script, which is a table of node values:

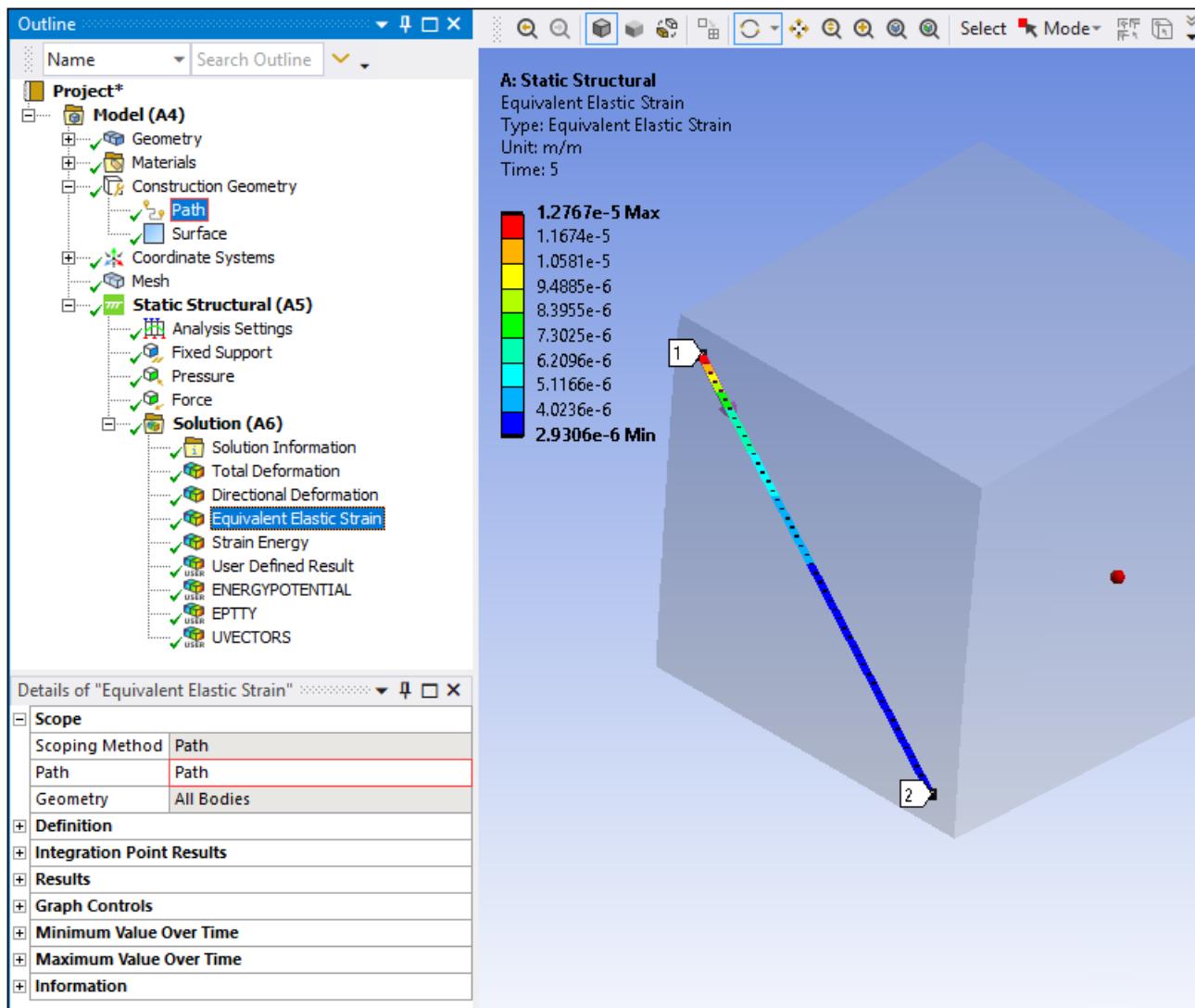
	Node	Values
		m
0	1671	-2.5766E-08
1	1702	-2.6568E-08
2	1733	-2.1311E-07
3	7227	-2.277E-07

An orange "">>>> |" prompt is visible at the bottom of the shell window.

Accessing Contour Results Scoped to Paths

Here is an example of a result scoped to a [Path](#).

Results



The result is scoped to the X, Y, and Z coordinates of the path sampling points, and the following code, once executed, displays them in tabular format.

The screenshot shows the Mechanical Scripting interface. At the top is a toolbar with icons for Editor, File, Save, Print, Run, Stop, and Help. Below the toolbar is a title bar "Mechanical Scripting". The main area has a tab labeled "New Script 1 : Description" containing the following Python script:

```
1 result := DataModel.GetObjectsByName("Equivalent Elastic Strain")[0]
2 resultTable := result.PlotData
3 print(resultTable)
4
```

Below the script editor is a "New Script 1" button and a "+" button. The bottom half of the window is a "Shell" window titled "Script Executed". It displays a table of data:

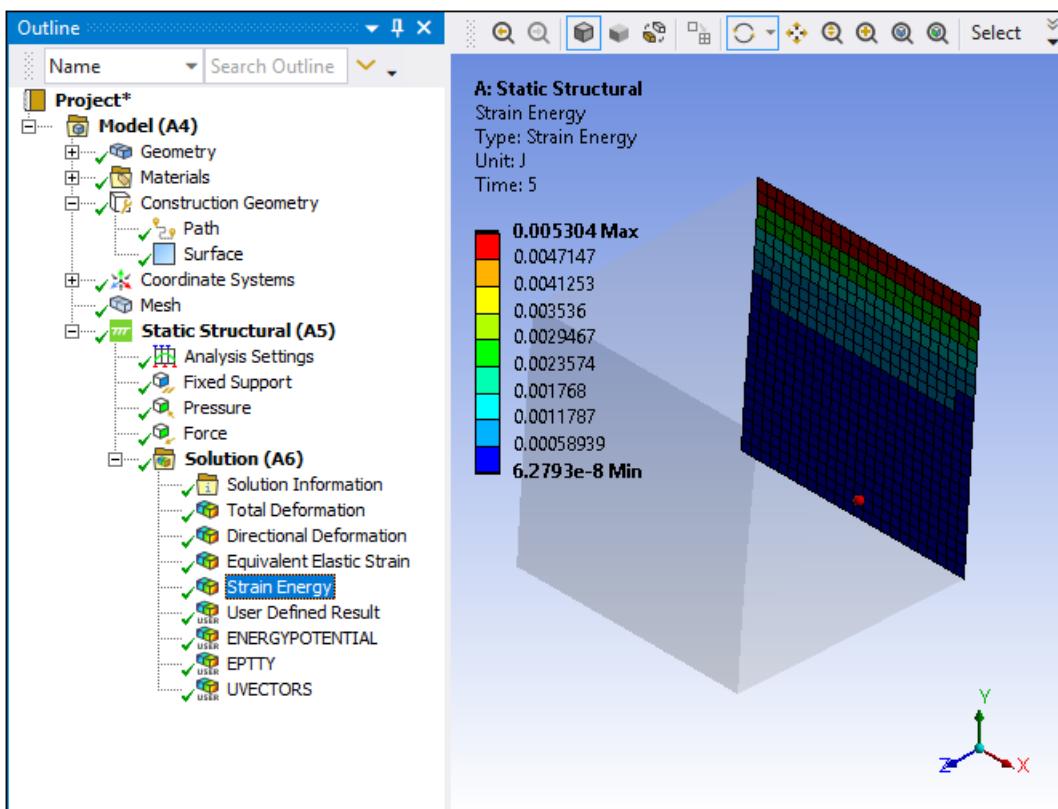
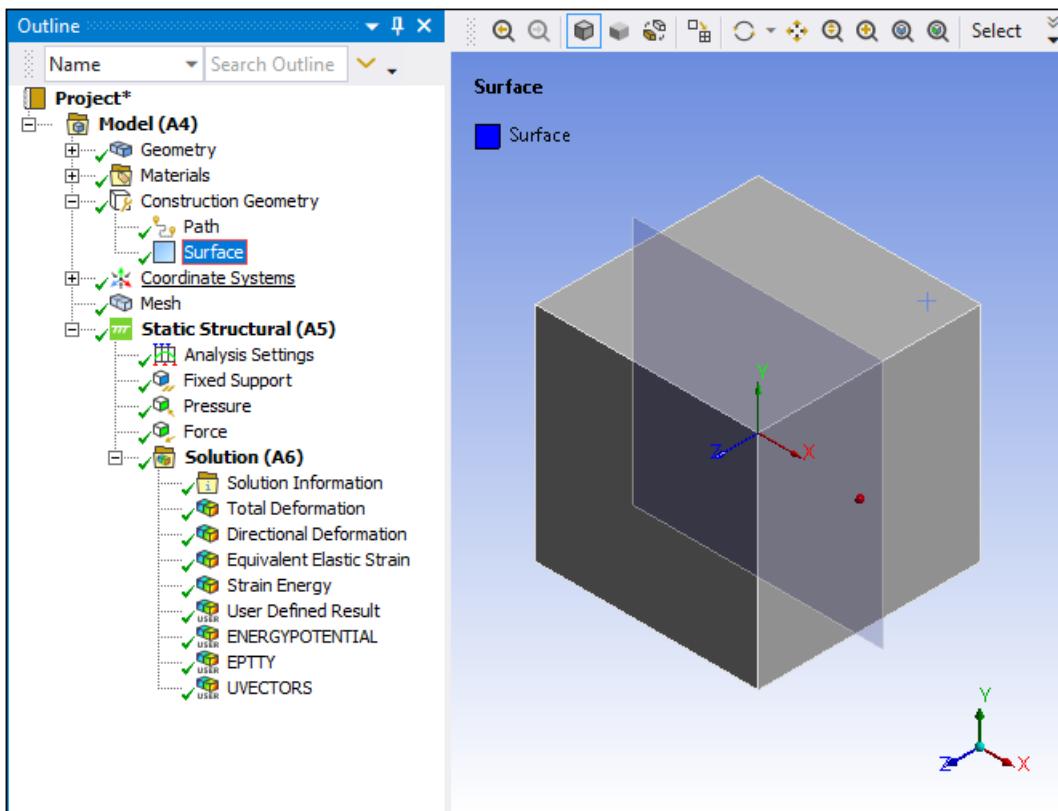
	X Coordi...	Y Coordi...	Z Coordi...	Values
	m/m			
0	0.083333	0.91667		1 1.2767E-05
1	0.10069	0.89931		1 1.1651E-05
2	0.11806	0.88194		1 1.0554E-05
...				
46	0.88194	0.11806		1 2.9642E-06
47	0.89931	0.10069		1 2.9732E-06
48	0.91667	0.083333		1 2.9802E-06

At the bottom left of the shell window is a red ">>>>" prompt.

Accessing Contour Results for Surfaces

You can create and evaluate on **Construction Geometry Surfaces**.

Results



For the Surface scoping, the following code displays the result in tabular format.

The screenshot shows the Mechanical Scripting interface. The top part is the 'Editor' window titled 'New Script 1 : Description'. It contains the following Python-like script:

```

1 result := DataModel.GetObjectsByName("Strain Energy")[0]
2 resultTable := result.PlotData
3 print(resultTable)
4

```

The bottom part is the 'Shell' window titled 'Script Executed'. It displays a table of results:

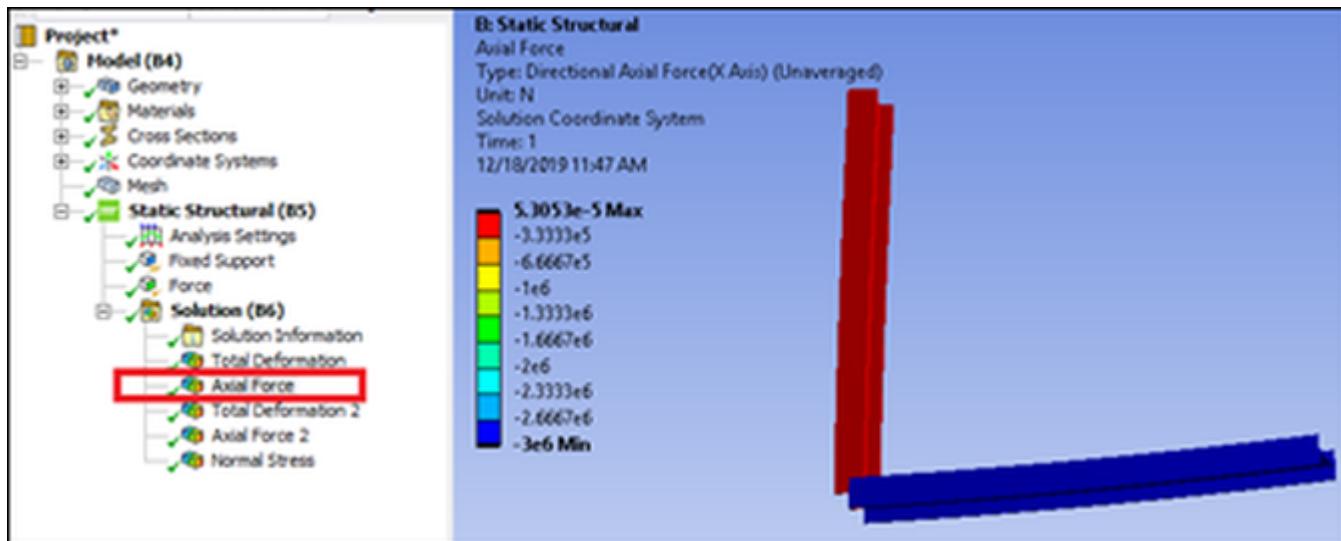
	X Coordi...	Y Coordi...	Z Coordi...	Element	Values
0	0.91667	0.91667	1E-05	1	0.005304
1	0.91667	0.95833	1E-05	1	0.005304
2	0.95833	0.95833	1E-05	1	0.005304
...					
2301	0	0.041667	1E-05	144	6.3011E-08
2302	0.041667	0.041667	1E-05	144	6.3011E-08
2303	0.041667	0	1E-05	144	6.3011E-08

Note:

PlotData displays result values along the X, Y and Z coordinates of the points on the surface. Exporting surface results displays the result values per facet that creates the surface and might have duplicate entries as facets share points. Hence the output of **PlotData** might miss the point or element pairs present when exporting the surface result.

Accessing Contour Results for Beams

For beam results, you can extract contour result values:



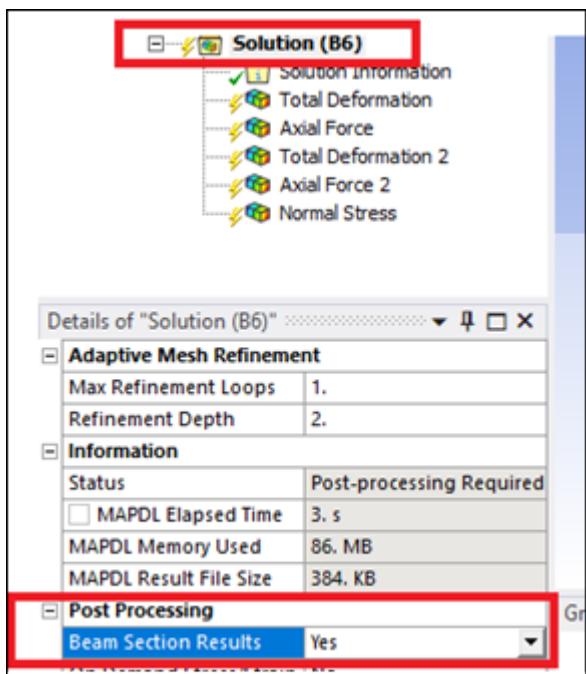
The following code accesses the evaluated contour results in the tabular data interface:

```
result = Model.Analyses[0].Solution.Children[2]
result.PlotData
```

The output looks like this:

	Node	Element	Values
N			
0	1	30	6.4191E-05
1	2	42	5.5148E-05
2	3	30	0.00023957
...			
84	28	56	-3000000
85	29	57	-3000000
86	30	58	-3000000

To view the beam cross section results, set **Solution** → **Beam Section Results** to Yes:



After reevaluating the result, using `PlotData` to access the beam cross section result gets multiple results across a single node:

	Node	Element	Values	
			N	
0	1	30	6.4191E-05	
1	1	30	6.4191E-05	
2	1	30	6.4191E-05	
...				
1041	30	58	-3000000	
1042	15	58	-3000000	
1043	15	58	-3000000	

See the following sections for additional examples:

- Access Contour Results for an Evaluated Result (p. 186)
- Write Contour Results to a Text File (p. 187)
- Access Contour Results at Individual Nodes/Elements (p. 187)

Limitations of Tabular Data Interface

With the tabular data interface, there are a few limitations:

- Results reflect how the application stores the result values, which means that they are not converted to the user-specified unit system.
- Path results do not display the length across the path of the different points of the path (S parameter). However, you can retrieve this information by exporting the results to a text file.

- If the path does not have results in all of the nodes (such as through a hole), the empty spaces currently display huge values (DBL_MAX).
- List slicing is not supported on column results.
- **PlotData** displays a snapshot of the result values stored in the application. If you change properties in the **Details** view, you must reevaluate and then call **PlotData** again to get the updated result values.

Set Result Display Options

You use the following command to access Result object display options (Preferences):

```
Graphics.ViewOptions.ResultPreference
```

The following properties are available with the **ResultPreference** object.

Property	Description
DeformationScaleMultiplier	Gets or sets the deformation scale multiplier for the result.
DeformationScaling	Gets or sets the deformation scale multiplier to either Auto Scale (DeformationScaling.Auto) or True Scale (DeformationScaling.True).
ShowMinimum	Gets or sets whether to display the result minimum value annotation label.
ShowMaximum	Gets or sets whether to display the result maximum value annotation label.
GeometryView	Gets or sets the result geometry view.
ContourView	Gets or sets the result contour view.
ExtraModelDisplay	Gets or sets the result edge display option.
IsoSurfaceValue	Gets or sets the capping value for Capped ISO surface view.
CappingType	Gets or sets the result capping type.
ScopingDisplay	Gets or sets the result scoping display.

Examples

Example 1

The following code sets the **DeformationScaleMultiplier** to 10 and the **DeformationScaling** to **True**.

```
Graphics.ViewOptions.ResultPreference.DeformationScaleMultiplier = 10
Graphics.ViewOptions.ResultPreference.DeformationScaling = MechanicalEnums.Graphics.DeformationScaling.True
```

Or...

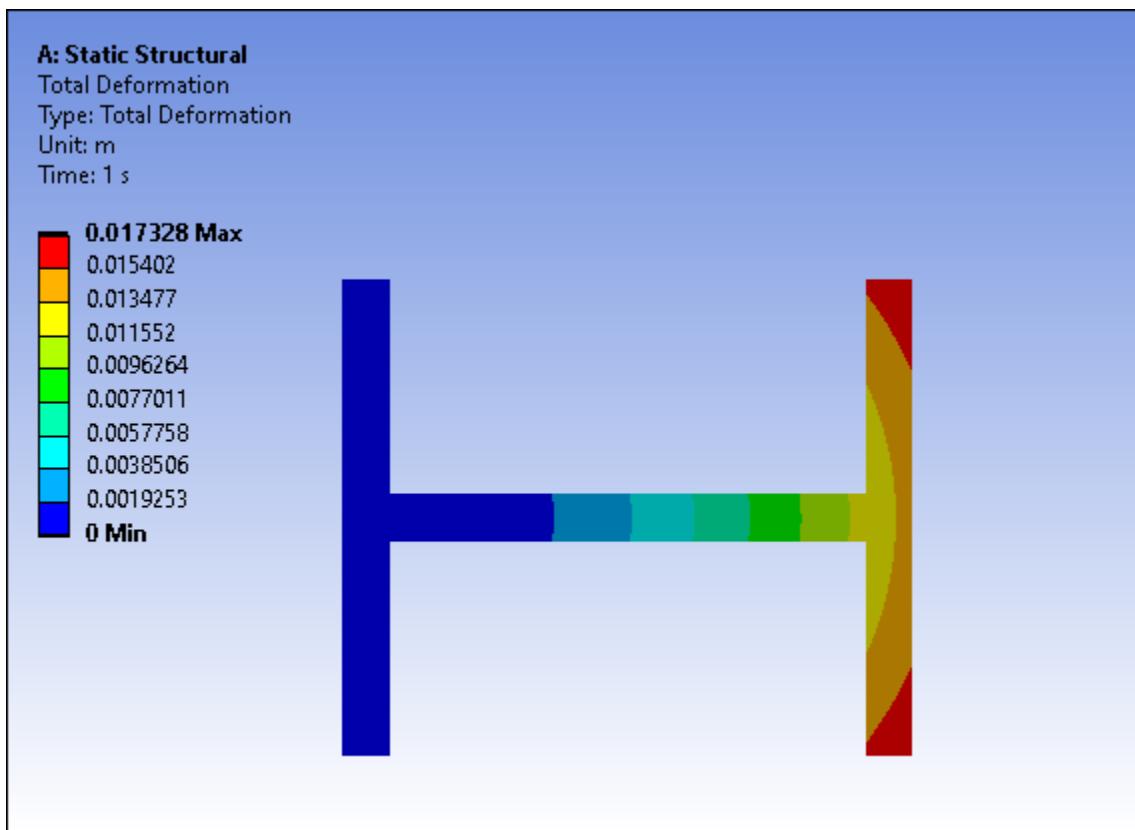
```
Graphics.ViewOptions.ResultPreference.DeformationScaleMultiplier = 10
Graphics.ViewOptions.ResultPreference.DeformationScaling = MechanicalEnums.Graphics.DeformationScaling.Auto
```

Note that the deformation Scale Factor has two options:

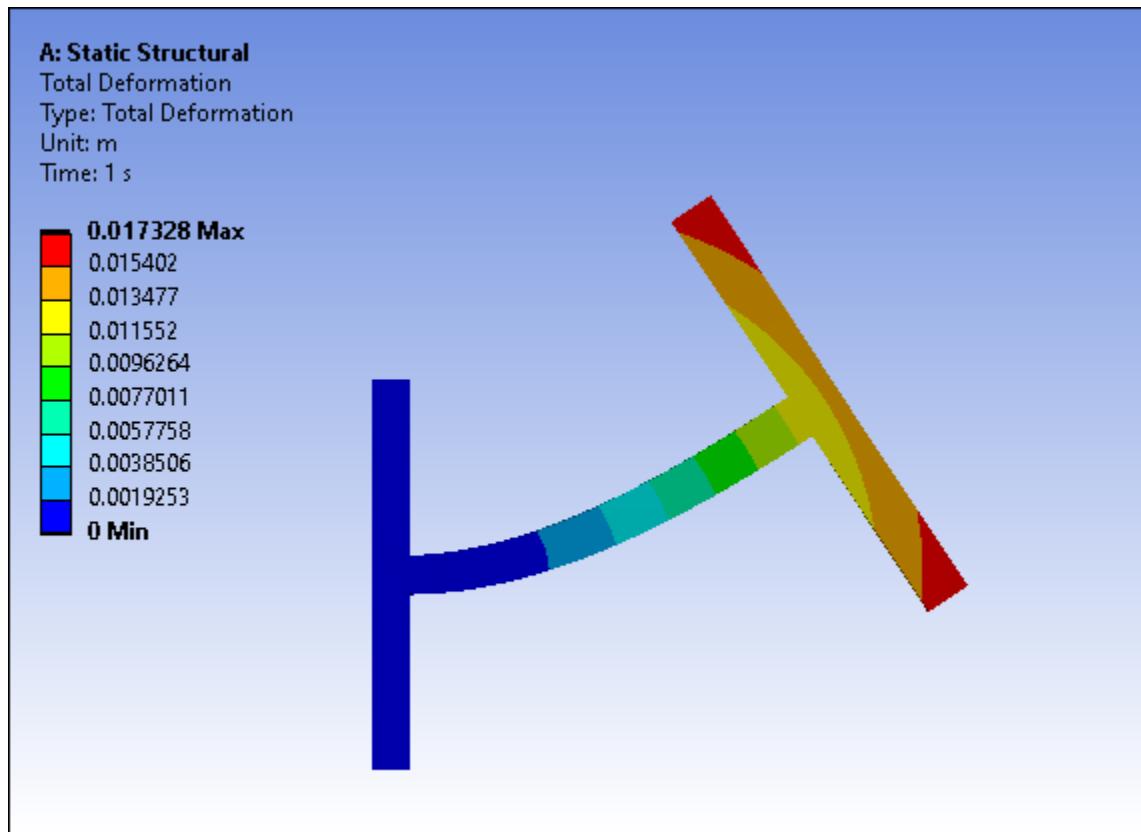
- **True** (True Scale): A multiplier for the actual deformation of the model.
- **Auto** (Auto Scale): A multiplier for an application controlled computation for an Auto Scale of the model.

The example deformations illustrated below demonstrate how an undeformed model is deformed with the True Scale and an Auto Scale factors.

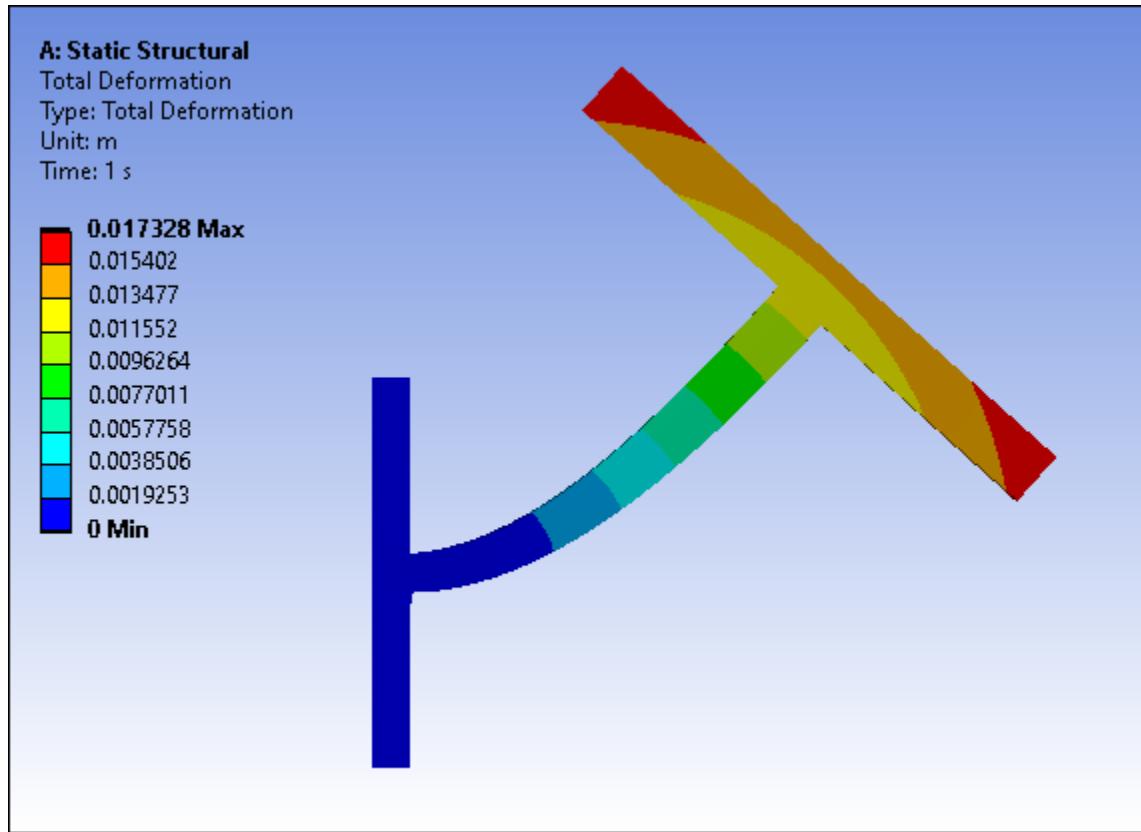
Undeformed



True Scale



Auto Scale

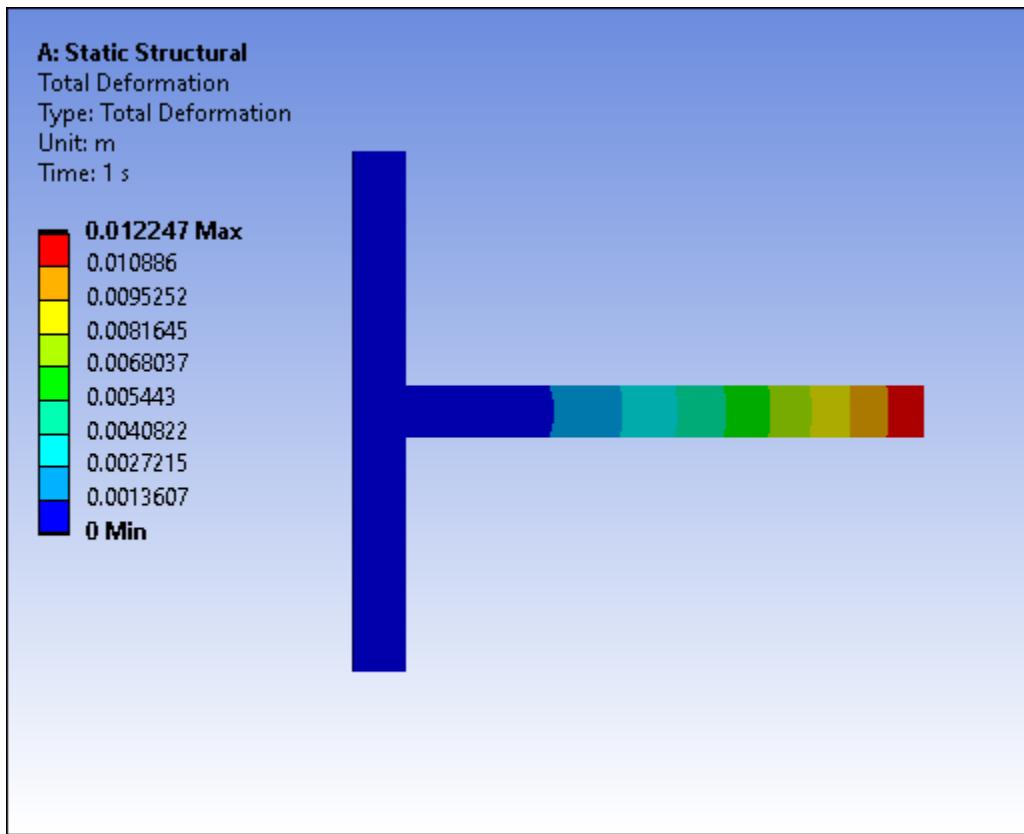


Example 2

The following code sets the:

- **GeometryView** property to **CappedIsoSurface**
- **IsoSurfaceValue** property to **0.013477**
- **CappingType** property to **Top**.

```
Graphics.ViewOptions.ResultPreference.GeometryView = MechanicalEnums.Graphics.GeometryView.CappedIsosurface
Graphics.ViewOptions.ResultPreference.IsoSurfaceValue = Quantity('0.013477')
Graphics.ViewOptions.ResultPreference.CappingType = MechanicalEnums.Graphics.CappingType.Top
```

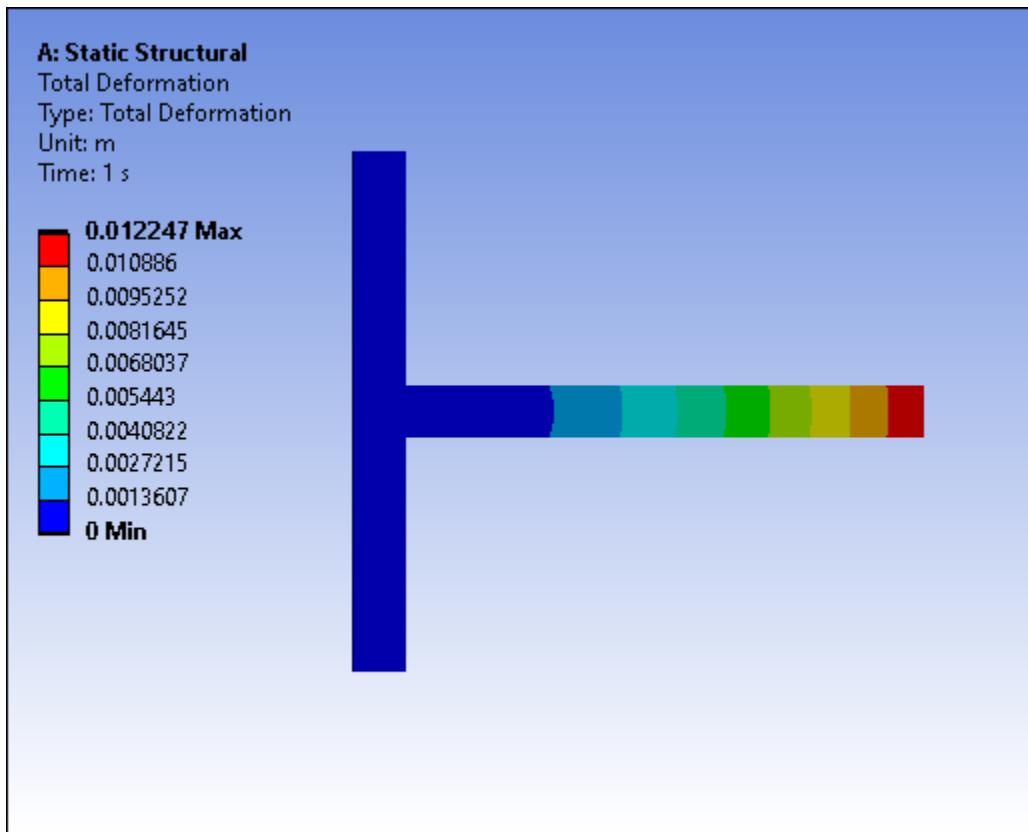


Example 3

When you scope the result to the bodies shown below, the following code sets the:

- **ContourView** property to **SmoothContours**
- **ExtraModelDisplay** property to **NoWireframe**
- **ScopingDisplay** property to **ResultOnly**

```
Graphics.ViewOptions.ResultPreference.ContourView = MechanicalEnums.Graphics.ContourView.SmoothContours
Graphics.ViewOptions.ResultPreference.ExtraModelDisplay = MechanicalEnums.Graphics.ExtraModelDisplay.NoWireframe
Graphics.ViewOptions.ResultPreference.ScopingDisplay = MechanicalEnums.Graphics.ScopingDisplay.ResultOnly
```



Working with Legends

Mechanical results are accompanied by legends. The properties that affect the behavior and appearance of a legend are referred to as legend settings. Legend settings can be either global (common to all results) or result-specific.

You can use legend settings objects to perform such operations as:

- Modifying the behavior and appearance of a legend shown on the screen
- Constructing an independent legend settings object and apply it to the current result
- Copying legend settings from one result object and apply it to another result

See the following sections for more information:

[Global Legend Settings](#)

[Result-Specific Legend Settings](#)

[Standalone Legend Settings](#)

Global Legend Settings

Legend settings that are common to all results are referred to as global legend settings. To access these settings, you use the command [GlobalLegendSettings](#):

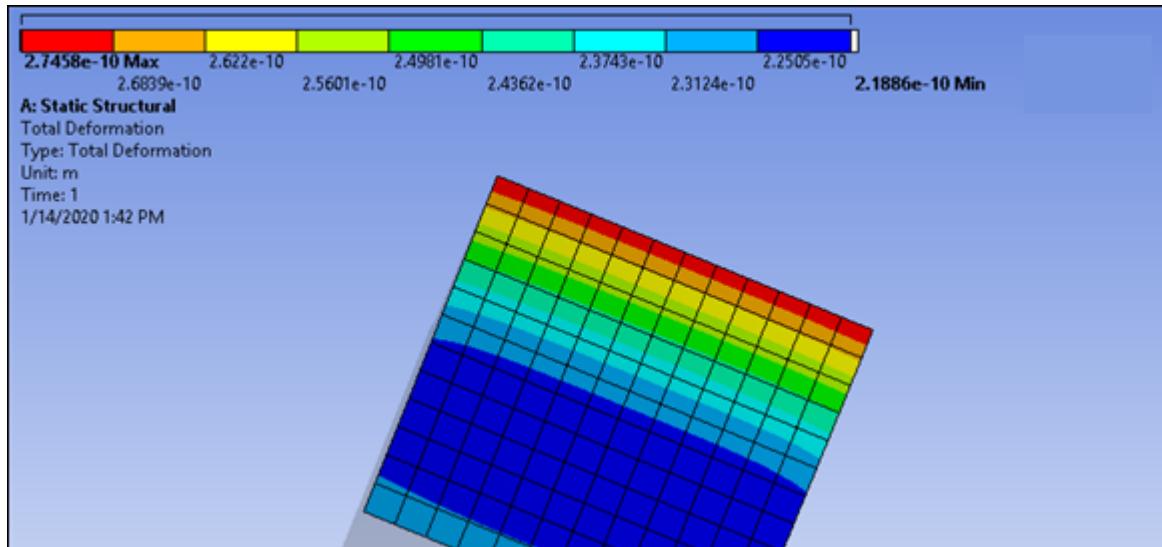
```
Graphics.GlobalLegendSettings
```

These properties are available for the **GlobalLegendSettings** object:

Property	Description
LegendOrientation	Gets or sets the legend orientation.
ShowDateAndTime	Gets or sets whether the date and time is shown.
ShowMinMax	Gets or sets whether the minimum and maximum values are shown.
ShowDeformingScaling	Gets or sets whether the deformation scaling is shown.
Reset	Resets the global legend settings to the defaults.

This code changes the legend orientation to horizontal and hides the deformation scaling:

```
gls = Graphics.GlobalLegendSettings
gls.LegendOrientation = LegendOrientationType.Horizontal
gls.ShowDateAndTime = True
gls.ShowMinMax = True
gls.ShowDeformingScaling = False
```



Result-Specific Legend Settings

Some legend settings are specific to the result object. To access result-specific legend settings, you first navigate to the result with the legend that you want to manipulate and then use the command **CurrentLegendSettings**:

```
Ansys.Mechanical.Graphics.Tools.CurrentLegendSettings()
```

Note:

You can only manipulate the legend of the result that is currently active.

These properties are available for the object for result-specific legend settings:

Property	Description
NumberOfBands	Gets or sets the number of bands on the legend.
AllScientificNotation	Gets or sets whether result values are to display in scientific notation.
Digits	Gets or sets the number of significant digits.
ColorScheme	Gets or sets the color scheme for the legend.
SemiTransparency	Gets or sets whether the legend is semi-transparent.
LogarithmicScale	Gets or sets whether the result values are distributed in a logarithmic scale.
HighFidelity	Gets or sets whether the high fidelity mode is on.
GetLowerBound	Gets the lower bound value of the specified band.
SetLowerBound	Sets the lower bound value of the specified band.
GetUpperBound	Gets the upper bound value of the specified band.
SetUpperBound	Sets the upper bound value of the specified band.
GetBandColor	Gets the color of the specified band.
SetBandColor	Sets the color of the specified band.
GetBandColorAuto	Gets whether the color of the specified band is set to Automatic .
SetBandColorAuto	Sets the color of the specified band to Automatic .
ResetColors	Resets all colors to default values.
ExportLegend	Exports the legend settings to an XML file.
Reset	Resets all legend customizations to default values.

This code modifies many result-specific legend settings for the current result object, including whether result values are to display in scientific notation, number of bands, color scheme, and semi-transparency, and it then exports these settings to an XML file:

```
legendSettings = Ansys.Mechanical.Graphics.Tools.CurrentLegendSettings()
legendSettings.AllScientificNotation = False
legendSettings.Digits = 4
legendSettings.NumberOfBands = 7
legendSettings.ColorScheme = LegendColorSchemeType.ReverseGrayScale
legendSettings.SemiTransparency = True
legendSettings.SetLowerBound(0, Quantity(2.38e-8, 'm'))
legendSettings.SetUpperBound(0, Quantity(3.38e-8, 'm'))
legendSettings.SetUpperBound(1, Quantity(4.38e-8, 'm'))
legendSettings.SetUpperBound(2, Quantity(5.38e-8, 'm'))
legendSettings.SetBandColor(0, Ansys.Mechanical.DataModel.Constants.Colors.Gray)
legendSettings.SetBandColor(1, Ansys.Mechanical.DataModel.Constants.Colors.Blue)
legendSettings.SetBandColor(2, Ansys.Mechanical.DataModel.Constants.Colors.Cyan)
legendSettings.SetBandColor(3, Ansys.Mechanical.DataModel.Constants.Colors.Green)
```

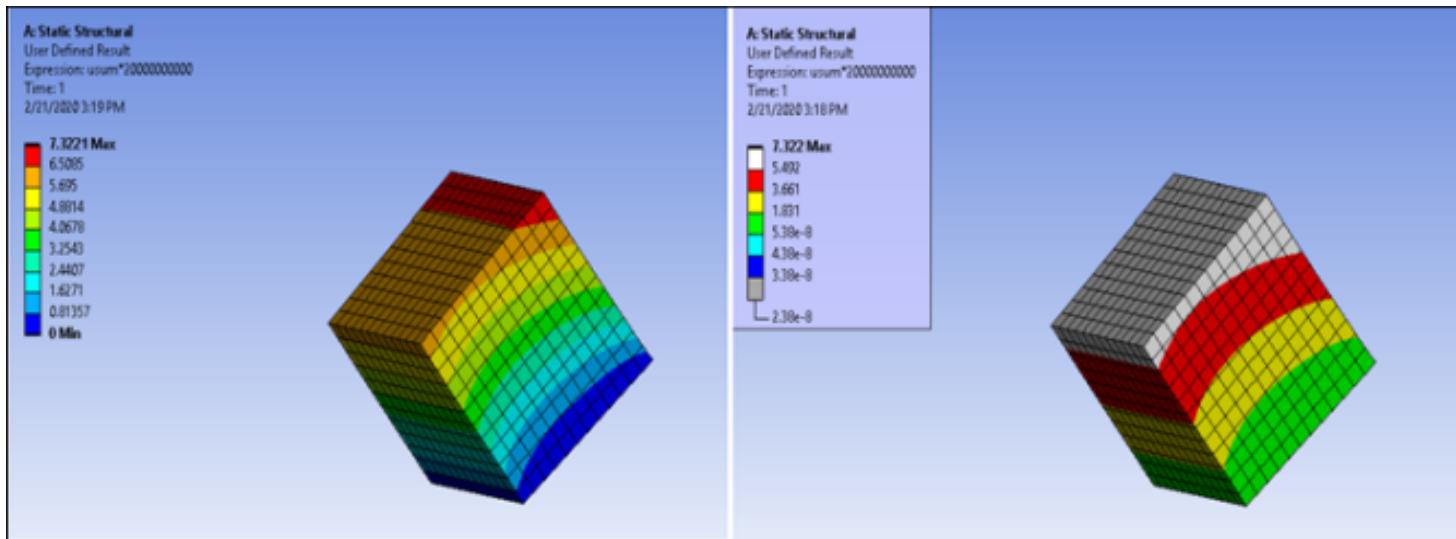
```
legendSettings.SetBandColor(4, Ansys.Mechanical.DataModel.Constants.Colors.Yellow)
legendSettings.SetBandColor(5, Ansys.Mechanical.DataModel.Constants.Colors.Red)
legendSettings.SetBandColor(6, Ansys.Mechanical.DataModel.Constants.Colors.White)
legendSettings.ExportLegend("E://file.xml")
```

Note:

Color can be set in RGB using **Ansys.Mechanical.DataModel.Utilities.Colors.RGB**:

```
legendSettings.SetBandColor(0,Ansys.Mechanical.DataModel.Utilities.Colors.RGB(255,0,0))
```

The following figure shows the legend using the original settings versus the modified settings.



Standalone Legend Settings

You can create an object for standalone legend settings, modify its properties, and then copy this object onto an existing result object.

To create the object with the standalone legend settings, you use the command [LegendSettings](#):

```
legendSettings = Ansys.Mechanical.Graphics.Tools.LegendSettings()
```

If you want to assign a default unit system, in the command, you would enter the unit system inside the parentheses:

```
legendSettings = Ansys.Mechanical.Graphics.Tools.LegendSettings("mm")
```

The object for the standalone legend settings has a default set of band values (from 0 to 9) and a default color scheme (Rainbow). You can set or modify any other properties that you care about. It is also possible to set the minimum and maximum for the standalone legend to mimic the minimum and maximum of a result:

```
legendSettings.SetMinMax(Quantity(1,'m'),Quantity(100,'m'))
```

If result-specific legend settings have been exported to an XML file, you can import these settings to create standalone legend settings:

```
legendSettings = Graphics.ImportLegend("E://file.xml", 'm')
```

Copying Legends

You can copy the legend settings from the current result onto an object for standalone legend settings. Conversely, you can create an object for standalone legend settings and then copy it onto a current result. These capabilities allow you to reuse a set of legend settings in various places.

These methods are available for copy operations:

Method	Description
MakeCopy()	Returns a standalone LegendSettings object that is a copy of the current result's legend.
CopyTo()	Copies a standalone LegendSettings object onto the current result's legend.
SetMinMax	Gets or sets whether the minimum and maximum values are shown.

Several examples demonstrate how to use these methods.

Example 1

This code creates a standalone **LegendSettings** object using the current result's legend:

```
# Navigate to a Total deformation object
totalDeform = DataModel.GetObjectsByName("Total Deformation")[0]
# Make a copy of its legend settings
totDefLegend = Ansys.Mechanical.Graphics.Tools.CurrentLegendSettings().MakeCopy()
```

Example 2

This code copies a standalone **LegendSettings** object onto the current result's legend:

```
# Create a standalone legend settings object
legendSetting = Ansys.Mechanical.Graphics.Tools.LegendSettings()
legendSetting.NumberOfBands = 3
legendSetting.SetBandColor(1, Ansys.Mechanical.DataModel.Constants.Colors.Red)

# Navigate to a result object
totalDeform = DataModel.GetObjectsByName("Total Deformation")[0]
totalDeform.Activate()

# Copy the standalone legend to this result
legendSetting.CopyTo(Ansys.Mechanical.Graphics.Tools.CurrentLegendSettings())
```

These copy operations can also be used for copying legends between different results.

Example 3

This code copies the legend from one result object onto another:

```
# Navigate to a result object
eqvStress = DataModel.GetObjectsByName("Equivalent Stress")[0]
eqvStress.Activate()

# Make a copy of its legend settings
eqvStressLegend = Ansys.Mechanical.Graphics.Tools.CurrentLegendSettings().MakeCopy()

# Navigate to a different result object
principalStress = DataModel.GetObjectsByName("Maximum Principal Stress")[0]
principalStress.Activate()

# Copy the previous legend to this result
eqvStressLegend.CopyTo(Ansys.Mechanical.Graphics.Tools.CurrentLegendSettings())
```


Other APIs

The following topics describe other Mechanical APIs of particular interest:

[Mechanical Interface and Ribbon Tab Manipulation](#)

[Command Snippets](#)

[Object Tags](#)

[Solve Process Settings](#)

[Message Window](#)

[Ray Casting on Geometry](#)

[Interacting with Legacy JScript](#)

[Data Processing Framework](#)

Mechanical Interface and Ribbon Tab Manipulation

[UserInterface](#) provides some control of the Mechanical user interface. This API is available using the following entry point:

`ExtAPI.UserInterface`

It allows you to control the ACT development, Button Editor, and ACT extension-defined ribbon tabs. It cannot be used to control other toolbars or to create new buttons, ribbon tabs, or toolbars.

`ExtAPI.UserInterface.Toolbars` is a collection of toolbar objects.

- Each object has fields such as `Name`, `Caption`, `Visibility`, and `child` to access entries.
- Each child has the following properties: `Caption`, `Enabled`, `Entries`, `EntryType`, `Name`, and `Visible`.

The Boolean fields `Visible` and `Enabled` can be set to `show` or `hide` so that you can control the availability of the buttons depending on the current context.

Command Snippets

[CommandSnippet](#) provides control of the `Commands` object. You use the method `AddCommandSnippet()` to insert a new child command snippet in the project tree:

```
sol = Model.Analyses[0].Solution
cs = sol.AddCommandSnippet()
cs.Input = "/COM, New input"
cs.AppendText("\n/POST1")
```

You can also use `ImportTextFile(string)` to import content from a text file or use `ExportTextFile(string)` to export a command snippet to a text file.

Object Tags

You can use APIs to access object tags and to add or remove them:

To access object tags:

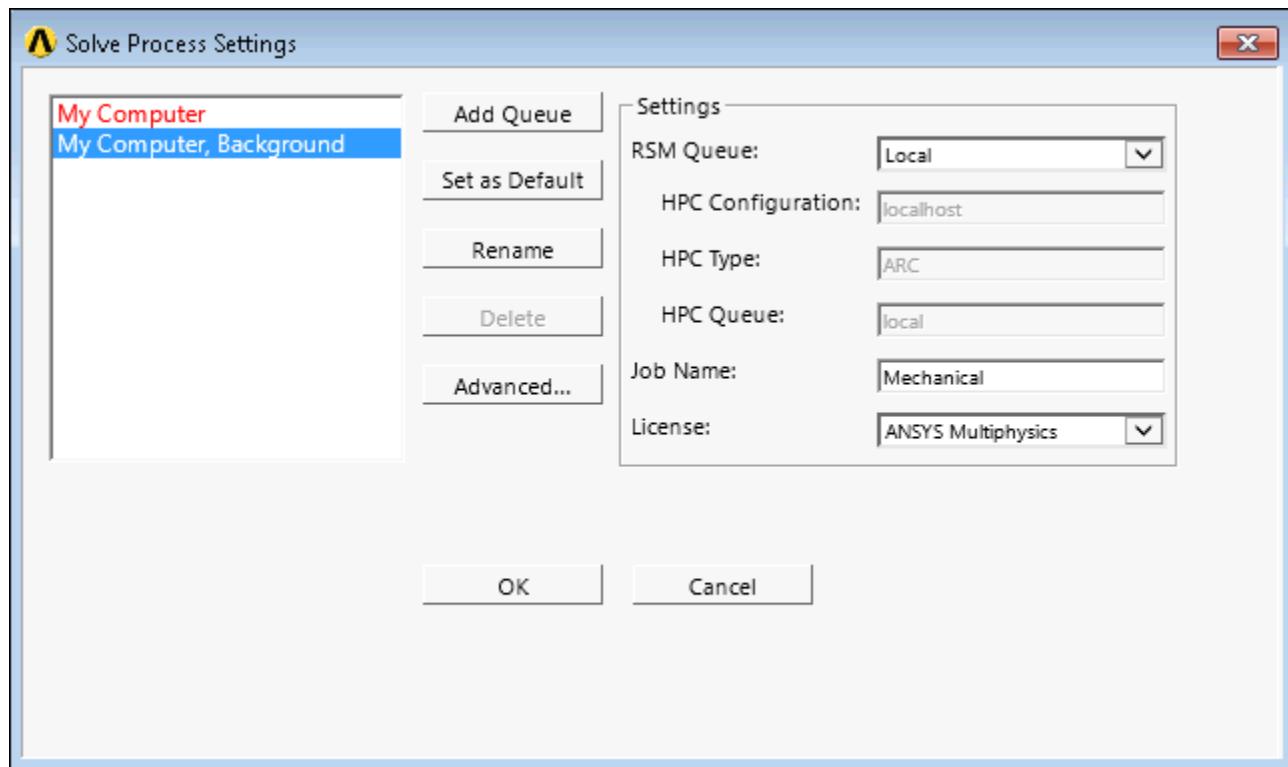
```
tag2 = DataModel.ObjectTags[2]
for tag in DataModel.ObjectTags:
    name = tag.Name
    objects = tag.Objects
```

To add or remove object tags:

```
tag = Ansys.Mechanical.Application.ObjectTag("supports")
tag.Objects = [obj1, obj2, obj3, ...]
DataModel.ObjectTags.Add(tag)
DataModel.ObjectTags.Remove( Ansys.Mechanical.Application.ObjectTag("loads") )
```

Solve Process Settings

The [Solve Process Settings](#) API is defined using two collections, a read-only collection of [RSMQueue](#) and a mutable collection of [SolveConfiguration](#). The [RSMQueue](#) collection lets you inspect the information on the right side of the **Solve Process Settings** dialog box for each possible selection in the **RSM Queue** property. Each item in the field on the left side is a [SolveConfiguration](#). Using the collection API, you can edit the collection or individual entities in the collection.



This script accesses and changes some details from the second solve configuration:

```
config2 = ExtAPI.Application.SolveConfigurations["My Computer, Background"]
x = config2.Default
y = config2.Settings.License
z = config2.SolveProcessSettings.ManualSolverMemorySettings.Workspace
config2.SolveProcessSettings.ManualLinuxSettings.UserName = "jane.doe"
config2.SolveProcessSettings.MaxNumberOfCores = 12
```

This script accesses some details from the first RSM queue:

```
queue1 = ExtAPI.Application.RSMQueues[0]
x = queue1.Name
y = queue1.HPCConfiguration
```

This script modifies the collection of solve configurations and solve:

```
collection = ExtAPI.Application.SolveConfigurations
new_config = Ansys.ACT.Mechanical.Application.SolveProcessSettings.SolveConfiguration()
new_config.Name = "My cluster"
collection.Add(new_config)
new_config.SetAsDefault()
new_config.Settings.License = "Mechanical Enterprise"
analysis.Solve()
```

Message Window

The Message window API uses a collection called [Messages](#). You can use this API to access individual messages and their data and operate on the collection to add and remove messages:

- Access a messages and its data:

```
thirdMsg = ExtAPI.Application.Messages[2]
for msg in ExtAPI.Application.Messages:
    obj = msg.Source
    stringId = msg.StringID
    caption = msg.DisplayString
    scoping = msg.Location
    time = msg.TimeStamp
    objs = msg.RelatedObjects
    severity = msg.Severity
```

- Add a message:

```
msg = Ansys.Mechanical.Application.Message("Problem!", MessageSeverityType.Error)
ExtAPI.Application.Messages.Add(msg)
```

Ray Casting on Geometry

A ray cast starts at a point and moves along a vector, finding geometry it intersects with along the path. With each intersection point, the normal of the intersected geometry can be determined. While Ansys uses this algorithm internally for many purposes, such as [Hit Point Normal axis selection](#) for coordinate systems, you can use it for your own purposes, such as defining coordinate systems.

To perform ray casting in Mechanical, use the method [CastRayOnGeometry\(\)](#), calling it like this:

```
Ansys.Mechanical.Selection.SelectionHelper.CastRayOnGeometry
```

The method **CastRayOnGeometry()** has the following inputs:

Method Input Name	Type	Functionality
castVector	BoundVector	Starting location and direction of the ray cast.
settings (Optional. Descriptions of defaults are provided later in this topic.)	GeometryRayCastSettings	Dictates how and what geometry entities should be hit and returned on the method call.

The inputs return a list of **GeometryRayCastHit** objects:

Method Output Name	Type	Functionality
HitVector	BoundVector	Starting location and unit direction on the geometry entity of the ray cast hit.
Entity	IGeoEntity	Geometry entity hit by ray cast.

GeometryRayCastSettings can be used to customize the behavior of the ray cast algorithm. It contains the following properties:

Setting Property Name	Type	Description
HitFaces	bool	Specifies if ray cast hits should return faces. Defaults to true.
HitEdges	bool	Specifies if ray cast hits should return edges. Defaults to false.
HitVertices	bool	Specifies if ray cast hits should return vertices. Defaults to false.
HitBodies	bool	Specifies if ray cast hits should return bodies. Defaults to false.
MaxHits	int	Specifies the maximum number of ray cast hits. Defaults to 1000.
CastRadius	Quantity	Specifies the maximum distance normal to the castVector qualifying as a hit. In other words, it is the cylinder around the castVector that intersects the geometry. Defaults to 1/500 the average of the overall bounding box.
CastLength	Quantity	Specifies the maximum distance along the castVector from the castVector origin. Defaults to go through all.

Example 1

This code creates a bound vector from which the ray cast originates and then casts it into the geometry as indicated earlier:

```
a_point = Point([2000,2000,2000], 'pm')
a_vector = Vector3D(-1,0,0)
a_bound_vector = Ansys.Mechanical.Math.BoundVector(a_point, a_vector)
```

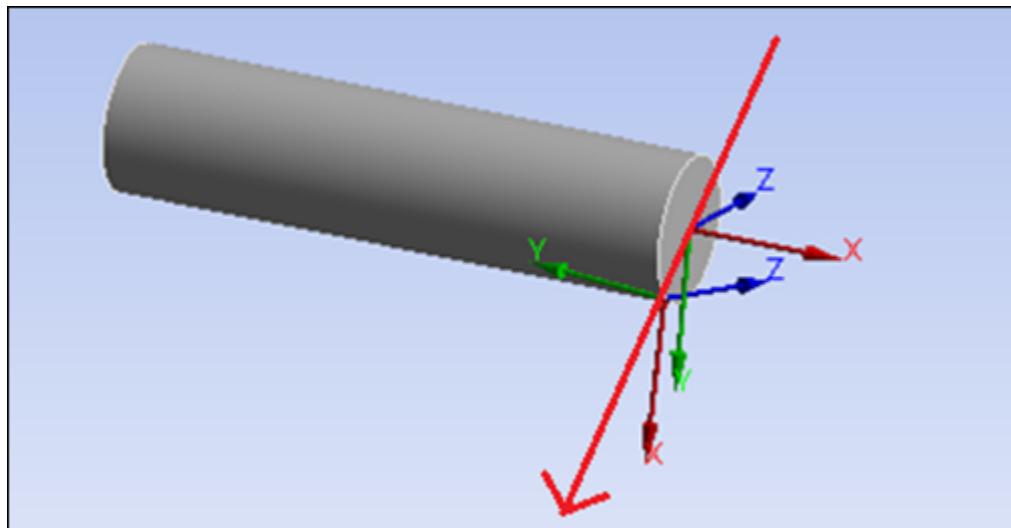
The result is used to create (or modify) a coordinate system by setting its **PrimaryAxisDirection**, **Origin**, and **OriginDefinedBy** to the output **BoundVector** to create a coordinate system for each geometry hit, visualizing the hit points and directions from the ray cast:

```
geom_hit = Ansys.Mechanical.Selection.SelectionHelper.CastRayOnGeometry(a_bound_vector)
for geom in geom_hit:
    print("\nNew geometry collision - " + str(geom.Entity.Type))
    print(geom.HitVector.Vector)
    print(geom.HitVector.Origin)

    cordSName = "Hitpoint " + str(list(geom_hit).index(geom))
    cordS = ExtAPI.DataModel.Project.Model.CoordinateSystems.AddCoordinateSystem()
    cordS.Name = cordSName

    cordS.PrimaryAxisDirection = geom.HitVector.Vector
    cordS.Origin = geom.HitVector.Origin.Location
    cordS.OriginDefineBy = CoordinateSystemAlignmentType.Fixe
```

The following images show output from the code:



```
New geometry collision - GeoFace
(0.258819043636322, 8.2512087289454E-08, 0.965925812721252)
Point((-0.000000007464,0.000000002,0.000000002), 'm')

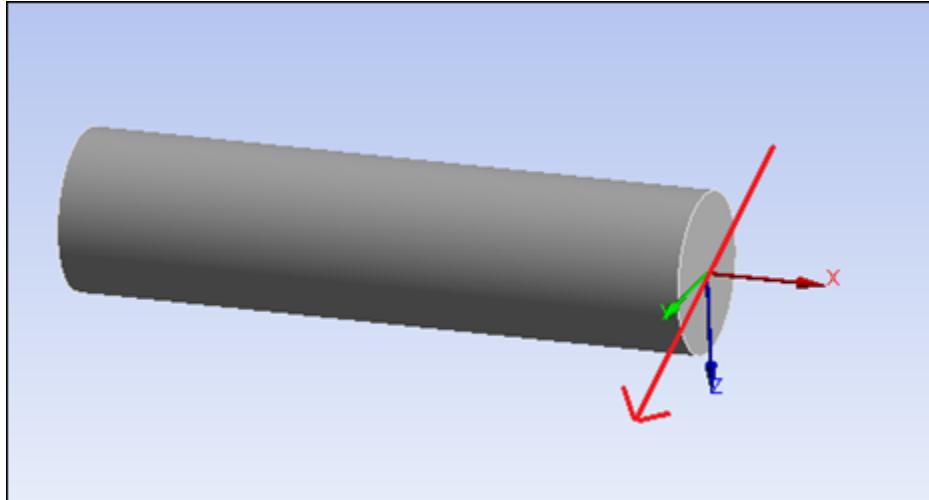
New geometry collision - GeoFace
(-0.963673889636993, 0.0682444721460342, 0.258215665817261)
Point((-0.004033320118,0.000000002,0.000000002), 'm')
```

Example 2

This code shows an example of changing ray cast settings, run with the same for loop in the second block as in the previous example:

```
a_point = Point([2000,2000,2000], 'pm')
a_vector = Vector3D(-1,0,0)
a_bound_vector = Ansys.Mechanical.Math.BoundVector(a_point, a_vector)
cast_settings = Ansys.Mechanical.Selection.GeometryRayCastSettings()
cast_settings.HitBodies = True
cast_settings.HitFaces = False
geom_hit = Ansys.Mechanical.Selection.SelectionHelper.CastRayOnGeometry(a_bound_vector, cast_settings)
```

The following images show output from the code:



```
New geometry collision - GeoBody  
(0.258819043636322, 8.25120878289454E-08, 0.965925812721252)  
Point((-0.00000007464,0.000000002,0.000000002), 'm')  
_
```

Interacting with Legacy JScript

When used from JScript that is executed from the Mechanical APIs, the JScript function **returnFromScript()** allows you return values from JScript back to Python or C#.

In the following code example, nothing is returned because **returnFromScript()** is not used:

```
x = ExtAPI.Application.ScriptByName( "jscript" ).ExecuteCommand( """var x=1""")
```

However, if you use the function **returnFromScript()**, you can return the value of x from the JScript back to the Python code:

```
x = ExtAPI.Application.ScriptByName( "jscript" ).ExecuteCommand( """var      x=1  
returnFromScript(x)""")
```

In this case, x now holds the value 1 because this value was passed into the Python code from the **returnFromScript()** function.

Supported Return Types

The function **returnFromScript()** can return values from the following data types:

- Int
- Double
- Boolean
- String

- Lists (Windows only)
- Dictionaries (Windows only) if created as Automation objects:

```
var dict = new ActiveXObject("Scripting.Dictionary");
```

Known Limitations

The following limitations exist:

- A list can hold a maximum of 65,535 items.
- Lists and dictionaries do not work on Linux.
- In the JScript that is to be executed using `ExecuteCommand()`, `ret` is a reserved keyword.

Example

The JScript that follows includes many functions for returning different types of variables:

```
script = """
    function returnInt(){
        return 10;
    }
Function returnDouble(){
    return 3.14;
}
function returnString(){
    return "Testing J Script";
}
function returnBool(){
    return true;
}
var innerDict = new ActiveXObject("Scripting.Dictionary");

function returnList(){
    var x = 10;
    var y = 20;
    var str = "thirty"
    var nullVal = null;
    var boolVal = true;
    var innerList1 = [1 ,2, 3, "str", [1, "str", 2]];
    var innerList2 = [[1], [1,2,3,[1,5,6]]];
    innerDict.Add("Testing", 1);

    var list = [x,y, str, nullVal, boolVal, innerList1,
innerList2, innerDict]
    return list;
}

function returnLongList(){
    var longList = [];
    for(var i= 0; i<65535; i++){
        longList[i] = i;
    }

    return longList;
}

function returnDict(){
    var dict = new ActiveXObject("Scripting.Dictionary");
    var listTry = [1,2,3];
    dict.Add("string", "strTest");
    dict.Add("int", 1);
}
```

```

        dict.Add("bool", true);
        dict.Add("null", null);
        dict.Add("list", [1,2,3]);
        dict.Add(1, "int");
        dict.Add(true, "bool");
        dict.Add([1,2,3], "list");

        innerDict.Add("Testing", 1);
        dict.Add("dict", innerDict);

        return dict;
    }
    var retVal = returnInt();
    returnFromScript(retVal );
}

"""
)

```

As demonstrated, you can set the variable **retVal** to whatever is returned from some function. You can then pass the variable **retVal** into **returnFromScript()** to return it to the Python code.

Return a list

The function **returnList()** can be used as a reference when returning lists from the JScript code.

Return a dictionary

The function **returnDict()** can be used as a reference when returning dictionaries from the JScript code.

Passing in a value to the JScript code from Python code

To pass in a value to the JScript code from Python code, you embed string values of Python code in the script.

The goal of this first example is to pass an integer value (5) to the JScript code from the Python code and then increment it by 1 in the JScript code and return the new value (6) to the Python code.

```

x = 5
script = """
var x = """ + str(x) + """;
x++;
returnFromScript(x);
"""
x = ExtAPI.Application.ScriptByName("jscript").ExecuteCommand(script)

```

The goal of this second example is to pass a list (1, 2, 3, 5) to the JScript code from the Python code and then update the fourth element and return the updated list (1, 2, 3, 10) to the Python code.

```

x = [1, 2, 3, 5]
script = """
var x = """ + str(x) + """;
x[3] = 10;
returnFromScript(x);

```

```
"""
x = ExtAPI.Application.ScriptByName("jscript").ExecuteCommand(script)
```

Note:

To pass in a Boolean value from Python to JScript, you must first convert it to a string and then make the string all lowercase because Booleans in Python start with an uppercase character where in JScript they start with a lowercase character.

Data Processing Framework

Introduction

The Data Processing Framework (DPF) is a tool that enables you to access and transform simulation data using customizable workflows. In the ACT console, with the IPython API of DPF, you can load results from Mechanical systems, apply post processing, and access data. DPF concepts:

- The data is contained in an entity called a **Field**. Fields can be grouped into a Fields Container to describe an analysis, such as "time steps for transient" or "harmonic analysis."
- The data is transformed by **Operators** that can be combined to define Workflows. Each Operator applies an elementary operation on the data.

DPF documentation is available in the ACT console. Available Operators with their descriptions and the required inputs/outputs are listed in this documentation. See the Available Operators menu. The documentation describes the entities used to contain the data (see Main data types) and the APIs available to explore the data with examples (see APIs).

Load the Framework

Open the **Mechanical Scripting** pane and import the Ans.DataProcessing module by running the following:

```
import mech_dpf
import Ans.DataProcessing as dpf
```

Generate Documentation

Use the following function to generate DPF's html documentation. This documentation contains examples and the library of Operators.

```
mech_dpf.help()
```

This function returns the path of the generated documentation. Copy and paste the path into a browser to view the documentation.

Access Data from Mechanical

You can use accessed data as input for DPF's Operators. To access data from the current Mechanical session, run the following:

```
mech_dpf.setExtAPI(ExtAPI)
```

Access Results

Run the following to obtain result data:

```
my_data_sources = mech_dpf.GetDataSources()
```

Access Mesh Data

Select a geometric entity in the Geometry and run the following to display the node and/or element IDs for the selected geometry:

```
my_nodes_scoping = mech_dpf.GetNodesScoping()
my_elements_scoping = mech_dpf.GetElementScoping()
```

Examples

This example shows how to read a displacement field on a selected geometry at a given time step and computes its normal.

```
import mech_dpf
import Ans.DataProcessing as dpf
mech_dpf.help()
mech_dpf.setExtAPI(ExtAPI)
my_data_sources = mech_dpf.GetDataSources()

#once a geometry is selected on the interface
my_nodes_scoping = mech_dpf.GetNodesScoping()
my_time_scoping = dpf.data.Scoping()
my_time_scoping.Ids = [1]

read_disp_op = dpf.operators.result.displacement()
read_disp_op.inputs.data_sources.Connect(my_data_sources)
read_disp_op.inputs.time_scoping.Connect(my_time_scoping)
read_disp_op.inputs.mesh_scoping.Connect(my_nodes_scoping)
norm_op = dpf.operators.math.norm_fc()
norm_op.inputs.fields_container.Connect(read_disp_op.outputs.fields_container)
my_u_norm_fc = norm_op.outputs.fields_container.GetData()
u_norm_1 = my_u_norm_fc.GetFieldByTimeId(1)

#see node ids of the field
u_norm_1.Scoping.Ids

#see displacement norm data
u_norm_1.Data
```

This example shows how to read a stress field on a selected geometry, average the elemental nodal stress on the nodes and compute its Von Mises equivalent.

```
import mech_dpf
import Ans.DataProcessing as dpf
mech_dpf.help()
mech_dpf.setExtAPI(ExtAPI)
my_data_sources = mech_dpf.GetDataSources()

#once a geometry is selected on the interface
my_elements_scoping = mech_dpf.GetElementScoping()
read_stress_op = dpf.operators.result.stress()
read_stress_op.inputs.data_sources.Connect(my_data_sources)
read_stress_op.inputs.mesh_scoping.Connect(my_elements_scoping)
```

```
# by default the result is read on one time set
average_op = dpf.operators.averaging.to_nodal_fc()
average_op.inputs.fields_container.Connect(read_stress_op.outputs.fields_container)
eqv_op = dpf.operators.invariant.von_mises_eqv_fc()
eqv_op.inputs.fields_container.Connect(average_op.outputs.fields_container)
my_stress_eqv_fc = eqv_op.outputs.fields_container.GetData()
my_stress_eqv1 = my_stress_eqv_fc.GetFieldByTimeId(1)

#see node ids of the field
my_stress_eqv1.Scoping.Ids

#see displacement norm data
my_stress_eqv1.Data
```

Scripting Examples

This section provides many examples of scripts that you can use to easily complete both common and novel tasks in Mechanical. Scripts are organized by three main categories, though some scripts overlap categories.

When logged into your Ansys customer account, you can search solutions for additional script examples using the following filter selections:

- For **Product**, select **ACT Customization Suite**.
- For **Product Family**, select **Scripting**.

To further limit the solutions shown, you can use the search box at the top of the page.

[Script Examples for Selection](#)

[Script Examples for Interacting with Tree Objects](#)

[Script Examples for Interacting with the Mechanical Session](#)

[Script Examples for Interacting with Results](#)

[Examples Using the Python Code Object](#)

[End-to-End Analysis Examples](#)

Script Examples for Selection

The following scripts are for making selections in Mechanical:

- Select Geometry or Mesh in the Graphics Window
- Get Tree Object of a Body Corresponding to a Selected Body
- Get GeoData Body Corresponding to a Tree Object of a Body
- Query Mesh Information for Active Selection
- Use an Existing Graphics Selection on a Result Object
- Calculate Sum of Volume, Area, and Length of Scoped Entities
- Create a Named Selection from the Scoping of a Group of Objects
- Create a Named Selection that Selects All Faces at a Specified Location
- Rescope a Solved Result Based on the Active Node or Element Selection
- Scope a Boundary Condition to a Named Selection
- Add a Joint Based on Proximity of Two Named Selections
- Print Selected Element Faces
- Get Normal of a Face
- Create a Selection Based on the Location of Nodes in Y
- Create Aligned Coordinate Systems in a Motor

Select Geometry or Mesh in the Graphics Window

Goal: Select a geometry or mesh in the graphics window.

Code for Geometry selection:

```
# Clear the current selection and select some previously determined Geo IDs
ExtAPI.SelectionManager.ClearSelection()
mySel = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
mySel.Ids = [17,19,22]    #These are the IDs of any geometric entities
ExtAPI.SelectionManager.NewSelection(mySel)
```

Code for Mesh selection:

You might need to be in the wireframe mode to see the selected nodes.

```
# Clear the current selection and select some mesh nodes
ExtAPI.SelectionManager.ClearSelection()
mySel = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.MeshNodes)
mySel.Ids = [1,2,3,4]    # These are the IDs of any node entities
ExtAPI.SelectionManager.NewSelection(mySel)
```

Get Tree Object of a Body Corresponding to a Selected Body

Goal: Get the tree object corresponding to a selected body in the graphics window.

Code:

```
myIds = ExtAPI.SelectionManager.CurrentSelection.Ids
geoBody = DataModel.GeoData.GeoEntityById(myIds[0])
treeBody = Model.Geometry.GetBody(geoBody)
```

Get GeoData Body Corresponding to a Tree Object of a Body

Goal: Get the GeoData (graphics window) body corresponding to a tree object of a body.

Code:

```
treeBody = Model.Geometry.Children[0].Children[0]
geoBody = treeBody.GetGeoBody()
```

Query Mesh Information for Active Selection

Goal: Query the mesh information for the active selection.

Code:

```
# Macro to demonstrate how to query mesh information for the active selection

meshData = DataModel.MeshDataByName("Global")
sel=ExtAPI.SelectionManager.CurrentSelection
refIds = sel.Ids

# check that current selection is nodes
if sel.SelectionType == SelectionTypeEnum.MeshNodes:
    print "%s Nodes Selected" % (refIds.Count)
    for meshId in refIds:
        node = meshData.NodeById(meshId)
        print "Node %s, X = %s, Y = %s, Z = %s" % (meshId, node.X, node.Y, node.Z)
else:
    print "Selection is not made of nodes"
```

Use an Existing Graphics Selection on a Result Object

Goal: Take an existing graphics selection and use it on a result object.

This example works best when selecting nodes and elements while already viewing a result.

Code:

```
mysel = ExtAPI.SelectionManager.CurrentSelection
myres = Tree.FirstActiveObject
myres.ClearGeneratedData()
myres.Location = mysel
myres.EvaluateAllResults()
```

Calculate Sum of Volume, Area, and Length of Scoped Entities

Goal: Calculate the sum of the volume, area, and length of scoped entities.

Code:

```
sum = 0

for geoid in ExtAPI.SelectionManager.CurrentSelection.Ids :
    geoEntity = DataModel.GeoData.GeoEntityById(geoid)
    if geoEntity.Type == GeoCellTypeEnum.GeoBody:
        sum += geoEntity.Volume
        type = "volume"
    if geoEntity.Type == GeoCellTypeEnum.GeoFace:
        sum += geoEntity.Area
        type = "area"
    if geoEntity.Type == GeoCellTypeEnum.GeoEdge:
        sum += geoEntity.Length
        type = "length"

# values are reported in the CAD unit system so get that
unit = Model.Geometry.LengthUnit
print("Total selected "+ type + " is: " + str(sum) + " " + str(unit))
```

Create a Named Selection from the Scoping of a Group of Objects

Goal: Loop over all contacts in the tree and create a single named selection scoped to all of the faces.

Code:

```
selmgr=ExtAPI.SelectionManager
selmgr.ClearSelection()
contacts = DataModel.GetObjectsByType(DataModelObjectCategory.ContactRegion)
for contact in contacts:
    selmgr.AddSelection(contact.SourceLocation)
    selmgr.AddSelection(contact.TargetLocation)

total=selmgr.CurrentSelection.Ids.Count

Model.AddNamedSelection()

print "Done with macro, Create a NS with %s selections" % (total)
```

Create a Named Selection that Selects All Faces at a Specified Location

Goal: Create a named selection that selects all faces at the zero location of the XY and XZ planes.

Code:

```
NS1 = DataModel.Project.Model.AddNamedSelection()
NS1.ScopingMethod = GeometryDefineByType.Worksheet
GenerationCriteria = NS1.GenerationCriteria
Criterion1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
Criterion1.Action = SelectionActionType.Add
Criterion1.EntityType = SelectionType.GeoFace
Criterion1.Criterion = SelectionCriterionType.LocationY
Criterion1.Operator = SelectionOperatorType.Equal
Criterion1.Value = Quantity("0 [m]")
GenerationCriteria.Add(Criterion1)
Criterion2 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
```

```
Criterion2.Action = SelectionActionType.Add
Criterion2.EntityType = SelectionType.GeoFace
Criterion2.Criterion = SelectionCriterionType.LocationZ
Criterion2.Operator = SelectionOperatorType.Equal
Criterion2.Value = Quantity("0 [m]")
GenerationCriteria.Add(Criterion2)
NS1.Generate()
```

Rescope a Solved Result Based on the Active Node or Element Selection

Goal: For a solved result, take the current selection in the graphics window (of either nodes or elements) and create a copy of the result scoped to this selection.

Code:

```
selmgr = ExtAPI.SelectionManager
loc = selmgr.CurrentSelection

res = Tree.FirstActiveObject

# verify object is a result or a custom result
isRegularResult = isinstance(res, Ansys.ACT.Automation.Mechanical.Results.Result)
isCustomResult = res.DataModelObjectCategory == DataModelObjectCategory.UserDefinedResult

if (isRegularResult or isCustomResult):

    newRes = res.Duplicate()
    newRes.ClearGeneratedData()
    newRes.Location=loc
    newRes.EvaluateAllResults()

else:
    print "Selected Object is not a Result!"
```

Scope a Boundary Condition to a Named Selection

Goal: Use a macro to create a fixed support and then scope it to a named selection.

Code:

```
# Macro to demonstrate the ability to create a fixed support and then scope it to a Named Selection
#
ns = DataModel.GetObjectsByName( "myFaces" )
# make sure only 1 named selection was found
if (ns.Count != 1):
    print("A single Named Selection was not Found!")
else:

    # get the first analysis and make sure it is structural
    analysis = DataModel.AnalysisList[0]

    if (analysis.PhysicsType != PhysicsType.Mechanical):
        print("The first analysis isn't structural")
    else:
        # Finally create a fixed support and scope it to the Named Selection
        mySupport = analysis.AddFixedSupport()
        mySupport.Location=ns[0]
        print("Done creating a Fixed Support and Scoping to a Named Selection")
```

Add a Joint Based on Proximity of Two Named Selections

Goal: Given a named selection of N faces and another of $N \times 2$ faces, add a joint between each face in the first named selection and the nearest two faces in the second named selection.

Code:

```

import math

#distance formula where c1 and c2 are each arrays of 3 real numbers
def dist(c1, c2):
    x = c1[0]-c2[0]
    x = x*x
    y = c1[1]-c2[1]
    y = y*y
    z = c1[2]-c2[2]
    z = z*z
    return math.sqrt(x+y+z)

jh1 = DataModel.GetObjectsByName("jh1")[0]           #named selection object of faces to become the single joint reference
jh2 = DataModel.GetObjectsByName("jh2")[0]           #named selection object of faces from which the two closest will be selected
group = DataModel.GetObjectsByName("Connection Group")[0]      #connection group to hold all of the joints that are created

geo = DataModel.GeoData
face_ids1 = jh1.Location.Ids           #face ids in jh1
face_ids2 = jh2.Location.Ids           #face ids in jh2

#get the centroids of all faces in face_ids2
face_centroids2 = [geo.GeoEntityById(face_id2).Centroid for face_id2 in face_ids2]

def get_min_indeces(face_centroid1):
    #array to hold the index and distance to face1_center for two closest faces in face_centroids2
    min_indeces = [None, None]

    for index in range(len(face_centroids2)):
        face_centroid2 = face_centroids2[index]
        distance = dist(face_centroid1, face_centroid2)

        #initialize with the first two iterations of this loop
        if index == 0:
            min_indeces[0] = (index, distance)
            continue
        if index == 1:
            min_indeces[1] = (index, distance)
            continue

        #get the index of the item with the larger distance
        if min_indeces[0][1] < min_indeces[1][1]:
            larger_dist_index = 1
        else:
            larger_dist_index = 0

        #replace that item with the current face if the distance is smaller
        if distance < min_indeces[larger_dist_index][1]:
            min_indeces[larger_dist_index] = (index, distance)
    return min_indeces

#use a transaction to speed up adding joints to the tree in a loop
with Transaction():
    for face_id1 in face_ids1:
        facel = geo.GeoEntityById(face_id1)
        face_centroid1 = facel.Centroid           #get the centroid of each face in face_ids1
        min_indeces = get_min_indeces(face_centroid1)

        #get the face ids using the indices computed above
        face_ids = [face_ids2[min_indeces[0][0]], face_ids2[min_indeces[1][0]]]

        #add a joint and assign its geometry selection

```

```

joint = group.AddJoint()

reference_selection = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
reference_selection.Ids = [face_id1]

mobile_selection = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
mobile_selection.Ids = face_ids

joint.ReferenceLocation = reference_selection
joint.MobileLocation = mobile_selection

```

Print Selected Element Faces

Goal: Print the element id and face index of all selected element faces.

Code:

```

# Purpose of the script: prints selected element faces area

g_elementTypeToElemFaceNodeIndices = {
    ElementTypeEnum.kTri3 : [ [ 0, 1, 2 ] ],
    ElementTypeEnum.kTri6 : [ [ 0, 1, 2, 4, 5, 6 ] ],

    ElementTypeEnum.kQuad4 : [ [ 0, 1, 2, 4 ] ],
    ElementTypeEnum.kQuad8 : [ [ 0, 1, 2, 4, 5, 6, 7, 8 ] ],

    ElementTypeEnum.kTet4 : [ [ 0, 1, 3 ], [ 1, 2, 3 ], [ 2, 0, 3 ], [ 0, 2, 1 ] ],
    ElementTypeEnum.kTet10 : [ [ 0, 1, 3, 4, 8, 7 ], [ 1, 2, 3, 5, 9, 8 ], [ 2, 0, 3, 6, 7, 9 ], [ 0, 2, 1, 6, 5 ],

    ElementTypeEnum.kPyramid5 : [ [ 0, 3, 2, 1 ], [ 0, 1, 4 ], [ 1, 2, 4 ], [ 3, 4, 2 ], [ 0, 4, 3 ] ],
    ElementTypeEnum.kPyramid13 : [ [ 0, 3, 2, 1, 8, 7, 6, 5 ], [ 0, 1, 4, 5, 10, 9 ], [ 1, 2, 4, 6, 11, 10 ], [ 3, 4, 5, 6, 12, 13 ] ],

    ElementTypeEnum.kWedge6 : [ [ 0, 2, 1 ], [ 3, 4, 5 ], [ 0, 1, 4, 3 ], [ 1, 2, 5, 4 ], [ 0, 3, 5, 2 ] ],
    ElementTypeEnum.kWedge15 : [ [ 0, 2, 1, 8, 7, 6 ], [ 3, 4, 5, 9, 10, 11 ], [ 0, 1, 4, 3, 6, 13, 9, 12 ], [ 1, 2, 3, 4, 5, 6, 14, 15 ] ],

    ElementTypeEnum.kHex8 : [ [ 0, 1, 5, 4 ], [ 1, 2, 6, 5 ], [ 2, 3, 7, 6 ], [ 3, 0, 4, 7 ], [ 0, 3, 2, 1 ], [ 4, 5, 6, 7 ] ],
    ElementTypeEnum.kHex20 : [ [ 0, 1, 5, 4, 8, 17, 12, 16 ], [ 1, 2, 6, 5, 9, 18, 13, 17 ], [ 2, 3, 7, 6, 10, 19, 11, 12 ] ]
}

def Norm(vec):
    return sqrt(vec[0]*vec[0] + vec[1]*vec[1] + vec[2]*vec[2])

def CrossProduct(a, b):
    return [ a[1]*b[2] - a[2]*b[1], a[2]*b[0] - a[0]*b[2], a[0]*b[1] - a[1]*b[0] ]

def TriangleArea(a, b, c):
    ab = [b[0]-a[0], b[1]-a[1], b[2]-a[2]]
    ac = [c[0]-a[0], c[1]-a[1], c[2]-a[2]]
    n = CrossProduct(ab, ac)
    area = 0.5 * Norm(n)
    #print("a={0}, b={1}, c={2}, ab={3}, ac={4}, n={5}, area={6}".format(a, b, c, ab, ac, n, area))
    return area

def GetElementFaceNodes(element_id, element_face_index):
    mesh = ExtAPI.DataModel.MeshDataByName("Global")
    element = mesh.ElementById(element_id)
    element_node_indices = g_elementTypeToElemFaceNodeIndices[element.Type][element_face_index]
    return [element.Nodes[element_node_index] for element_node_index in element_node_indices]

def GetElementFaceArea(element_id, element_face_index):
    element_face_nodes = GetElementFaceNodes(element_id, element_face_index)
    if len(element_face_nodes) == 3 or len(element_face_nodes) == 6:
        # it's a triangle
        return \
            TriangleArea( \
                [element_face_nodes[0].X, element_face_nodes[0].Y, element_face_nodes[0].Z], \
                [element_face_nodes[1].X, element_face_nodes[1].Y, element_face_nodes[1].Z], \

```

```

        [element_face_nodes[2].X, element_face_nodes[2].Y, element_face_nodes[2].Z])
    else:
        # it's a quadrangle (made of 2 triangles)
        return \
            TriangleArea( \
                [element_face_nodes[0].X, element_face_nodes[0].Y, element_face_nodes[0].Z], \
                [element_face_nodes[1].X, element_face_nodes[1].Y, element_face_nodes[1].Z], \
                [element_face_nodes[2].X, element_face_nodes[2].Y, element_face_nodes[2].Z]) + \
            TriangleArea( \
                [element_face_nodes[0].X, element_face_nodes[0].Y, element_face_nodes[0].Z], \
                [element_face_nodes[2].X, element_face_nodes[2].Y, element_face_nodes[2].Z], \
                [element_face_nodes[3].X, element_face_nodes[3].Y, element_face_nodes[3].Z])

def GetCurrentSelectedElementFaces():
    current_selection = ExtAPI.SelectionManager.CurrentSelection
    if current_selection.SelectionType == SelectionTypeEnum.MeshElementFaces:
        element_ids = current_selection.Ids
        element_face_indices = current_selection.ElementFaceIndices
        return (element_ids, element_face_indices)
    elif current_selection.SelectionType == SelectionTypeEnum.MeshElements:
        element_ids = current_selection.Ids
        element_face_indices = [0 for i in element_ids]
        return (element_ids, element_face_indices)
    else:
        return ([], [])

def PrintCurrentSelectedElementFacesElementFacesArea():
    (element_ids, element_face_indices) = GetCurrentSelectedElementFaces()
    if len(element_ids) < 1:
        print('No element faces selected')
        return

    text = ''
    for i in range(len(element_ids)):
        area = GetElementFaceArea(element_ids[i], element_face_indices[i])
        text += 'element id={0}, faceIndex={1}, area={2}\n'.format(element_ids[i], element_face_indices[i], area)
    print(text)

PrintCurrentSelectedElementFacesElementFacesArea()

```

Get Normal of a Face

Goal: Print the normal of a given face at a given location in space.

Code:

```

face_id = 20 #Current selection in ExtAPI.SelectionManager.CurrentSelection
point = (.01,.015,0.) #point in xyz space in the CAD unit system

#get the face object
face = DataModel.GeoData.GeoEntityById(face_id)

#get the projected point on the geometry (a curvilinear abscissa for an edge, (u,v) for a face)
u,v = face.ParamAtPoint(point)

#print the normal at the point
print(face.NormalAtParam(param))

```

Create a Selection Based on the Location of Nodes in Y

Goal: Given a Y location and tolerance, create a selection of all nodes within that tolerance from their global Y coordinate location.

Code:

```
# Get access to mesh
mesh = DataModel.MeshDataByName(ExtAPI.DataModel.MeshDataNames[0])
selected_node_ids = set() # Empty set
for node in mesh.Nodes:
    tolerance = 0.0001 # 0.1 mm
    y_location = 0.0130
    if (node.Y >= y_location - tolerance) and (node.Y <= y_location + tolerance):
        selected_node_ids.add(node.Id)

# Create selection
selection_info = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.MeshNodes)
for selected_node_id in selected_node_ids:
    selection_info.Ids.Add(selected_node_id)

ExtAPI.SelectionManager.NewSelection(selection_info)
```

Create Aligned Coordinate Systems in a Motor

Goal: Given a motor model with a named selection of faces and two named selections of vertices, all of the same size, for each of the three lists, add a coordinate system at the center of the face with its +Y axis pointed out of the face. The -X direction will be from the vertex in the second list to the vertex in the third list.

Note:

The following script example is model-specific. Click to download the archived Ansys project[here](#).

Code:

```
# Get all faces and vertices of named selections
face_named_selection = Model.NamedSelections.Children[0] # First named selection, of faces
face_ids = face_named_selection.Ids
start_vertex_named_selection = Model.NamedSelections.Children[1] # Second named selection, of vertices
start_vertex_ids = start_vertex_named_selection.Ids
end_vertex_named_selection = Model.NamedSelections.Children[2] # Third named selection, of vertices
end_vertex_ids = end_vertex_named_selection.Ids

# Create axis systems
selection_info = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
with Transaction(): # Suppress tree update for performance
    for iFaceCount in range(0, face_ids.Length):
        coordinate_system = Model.CoordinateSystems.AddCoordinateSystem()
        selection_info.Ids.Clear()
        selection_info.Ids.Add(face_ids[iFaceCount]) # Create a temporary selection

        # Define origin in face center
        coordinate_system.OriginLocation = selection_info

        # Define primary axis (Y) normal into face
        coordinate_system.PrimaryAxis = CoordinateSystemAxisType.PositiveYAxis
        coordinate_system.PrimaryAxisDefineBy = CoordinateSystemAlignmentType.Associative # Geometry selection
        coordinate_system.PrimaryAxisLocation = selection_info

        # Define orientation around principle axis from start- to endvertex
        selection_info.Ids.Clear()
        selection_info.Ids.Add(end_vertex_ids[iFaceCount]) # Create a temporary selection
        selection_info.Ids.Add(start_vertex_ids[iFaceCount]) # Create a temporary selection
        coordinate_system.SecondaryAxis = CoordinateSystemAxisType.PositiveXAxis
```

```
coordinate_system.SecondaryAxisDefineBy = CoordinateSystemAlignmentType.Associative # Geometry selection  
coordinate_system.SecondaryAxisLocation = selection_info
```

Script Examples for Interacting with Tree Objects

The following scripts are for interacting with tree objects in Mechanical:

- Delete an Object
- Refresh the Tree
- Clear the Mesh
- Get All Visible Properties for a Tree Object
- Parametrize a Property for a Tree Object
- Create Construction Lines from Cylindrical Faces
- Update Geometries for all Construction Lines
- Create Material Assignment from Body Materials
- Count the Number of Contacts
- Suppress Duplicate Contacts
- Verify Contact Size
- Set Pinball to 5mm for all Frictionless Contacts
- Use a Named Selection as Scoping of a Load or Support
- Suppress Bodies Contained in a Given Named Selection
- Rename a Named Selection Based on Scoping
- Modify the Scoping on a Group of Objects
- Change Tabular Data Values of a Standard Load or Support
- Duplicate an Harmonic Result Object
- Retrieve Object Details Using SolverData APIs
- Evaluate Spring Reaction Forces
- Export a Result Object to an STL File
- Export Result Images to Files
- Tag and Group Result Objects Based on Scoping and Load Steps
- Work with Solution Combinations
- Create a Pressure Load
- Create a Convection Load
- Create Node Merge Object at a Symmetry Plane
- Access Contour Results for an Evaluated Result
- Write Contour Results to a Text File
- Access Contour Results at Individual Nodes/Elements
- Set Arbitrary Coordinate System Properties
- Transform Coordinate Systems (with Math)

[Select Objects By Name](#)

[Export Figures](#)

Delete an Object

Goal: Delete a selected object from the tree.

Code:

```
ObjToDelete.Delete()
```

Refresh the Tree

Goal: Refresh the Mechanical tree. This is needed for certain operations that visually appear only after an update of the tree.

Code:

```
Tree.Refresh()
```

Clear the Mesh

Goal

The following one-line script enables you to clear your generated mesh.

Code

```
# Clears generated mesh  
Model.Mesh.ClearGeneratedData()
```

Get All Visible Properties for a Tree Object

Goal: Get all visible properties for a native tree object and then print their captions and string values.

Code:

```
force = Model.Analyses[0].AddForce()  
for forceProperty in force.VisibleProperties:  
    print(forceProperty.Caption + " | " + forceProperty.StringValue)
```

Parametrize a Property for a Tree Object

Goal: Parametrize a **Details View** property for a native tree object and then print out the Workbench ID.

Code:

```
hydrostaticPressure = Model.Analyses[0].AddHydrostaticPressure()
fluidDensity = hydrostaticPressure.CreateParameter("FluidDensity")
fluidDensity.ID
```

Create Construction Lines from Cylindrical Faces

Goal

The following script example enables you to create a [Construction Line](#) using cylindrical faces and the axis to rotation (parallel to global Y) about those faces. This is essentially an abstraction of a bolt.

Code

```
# From a selection of cylindrical faces, create construction line on the cylinder's axis

curSel=ExtAPI.SelectionManager.CurrentSelection
construction_geometry = Model.ConstructionGeometry
if construction_geometry == None:
    construction_geometry = Model.AddConstructionGeometry()

for faceId in curSel.Ids:
    print faceId
    face=ExtAPI.DataModel.GeoData.GeoEntityById(faceId)
    if face.Type != GeoCellTypeEnum.GeoFace:
        continue;

    if face.SurfaceType != GeoSurfaceTypeEnum.GeoSurfaceCylinder:
        continue;

    mid1=face.Edges[0].Centroid
    mid2=face.Edges[1].Centroid
    construction_line = construction_geometry.AddConstructionLine()
    [global_points_1] = construction_line.CreatePoints([mid1])
    [global_points_2] = construction_line.CreatePoints([mid2])
    edge_collection = construction_line.CreateStraightLines([global_points_1, global_points_2], [(0, 1)])

    with Transaction():
        construction_line.AddToGeometry()
```

Update Geometries for all Construction Lines

Goal

The following script example enables you to update [Construction Line](#) geometries.

Code

```
# Automatically update all geometries of all construction lines

constructionLines=ExtAPI.DataModel.GetObjectsByType(Ansys.ACT.Automation.Mechanical.ConstructionLines.Construc
with Transaction():
    for constLine in constructionLines:
        constLine.UpdateGeometry()
```

Create Material Assignment from Body Materials

Goal

The following script example creates a [Material Assignment](#) object, specifies object properties, and assigns a material to all bodies of your model.

Code

```
# Clears generated mesh
# Single line script to be used to create a user button and possibly a keyboard shortcut

# Create dictionary with materials and corresponding bodies
bodies = DataModel.GetObjectsByType(DataModelObjectCategory.Body)
listMaterials={}
for body in bodies:
    if body.Material in listMaterials.keys():
        listMaterials[body.Material].append(body.ObjectID)
    else:
        listMaterials[body.Material]=[body.ObjectID]

# Now create (or update) material assignments
matFolder=ExtAPI.DataModel.GetObjectsByType(Ansys.ACT.Automation.Mechanical.Materials)[0]
# first find existing material assignment
existingMatAssg={}
for ch in matFolder.Children:
    if ch.GetType() == Ansys.ACT.Automation.Mechanical.MaterialAssignment:
        if ch.Material in existingMatAssg.keys():
            existingMatAssg[ch.Material].append(ch.ObjectID)
        else:
            existingMatAssg[ch.Material]=[ch.ObjectID]

# Now loop of body materials and create or update assignment
for mat in listMaterials.keys():
    # Select corresponding bodies
    ExtAPI.SelectionManager.ClearSelection()
    geoBodyList=[]
    temp_sel= ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
    for item in listMaterials[mat]:
        geoBodyList.append(ExtAPI.DataModel.GetObjectById(item).GetGeoBody().Id)
    temp_sel.Ids=geoBodyList
    ExtAPI.SelectionManager.NewSelection(temp_sel)
    if mat in existingMatAssg.keys(): # assignment already exist, update scoping
        assg=ExtAPI.DataModel.GetObjectById(existingMatAssg[mat][0])
        assg.Location=temp_sel
    else: # create new assignment
        newAssg=matFolder.AddMaterialAssignment()
        newAssg.Material=mat
        newAssg.Location=temp_sel
```

Count the Number of Contacts

Goal: Count the number of contacts in the model.

Code:

```
contacts = DataModel.GetObjectsByType(DataModelObjectCategory.ContactRegion)
numContacts = contacts.Count
print("There are %s contact regions" % (numContacts) )
```

Suppress Duplicate Contacts

Goal

The following script example examines your model to determine if duplicate contact pairs exist and if so, automatically suppresses the duplicate pair. A duplicate pair is a contact pair where the Source and Target scoping are interchangeable, that is, the same.

Code

```
# Scan contacts to find duplicate pairs.

# Get all contact regions
contacts=ExtAPI.DataModel.GetObjectsByType(DataModelObjectType.ContactRegion)

contLocation={}
duplicateCont={}

# create dictionary will all sources and locations from all contacts
for cont in contacts:
    l1=[]
    l2=[]
    for id in cont.SourceLocation.Ids:
        l1.append(id)
    for id in cont.TargetLocation.Ids:
        l2.append(id)
    contLocation[cont.ObjectId]=[l1,l2]

dupList=[]
# Now check for duplicates
for cont in sorted(contLocation.keys()):
    # Retrieve source and location
    source1=contLocation[cont][0]
    target1=contLocation[cont][1]
    for contTest in sorted(contLocation.keys()):
        if contTest != cont and contTest not in dupList: # Don't check contact with itself, don't check if already checked
            # Retrieve source and location for test contacts
            source2=contLocation[contTest][0]
            target2=contLocation[contTest][1]
            if (source1==source2 and target1==target2) or (source2==target1 and source1==target2):
                # Contacts are duplicate of each other, add to duplicateCont dictionary if not already identified
                if cont in duplicateCont.keys():
                    curList=duplicateCont[cont]
                    curList.append(contTest)
                    d1={cont: curList}
                    duplicateCont.update(d1)
                else:
                    duplicateCont[cont]=[contTest]
            # This list contains all contacts already identified as duplicate to avoid double detection
            dupList.append(contTest)
            dupList.append(cont)

msgCont=''
# Prepare message with list of duplicate contacts and suppress duplicated ones
for cont in duplicateCont.keys():
    cont1=ExtAPI.DataModel.GetObjectById(cont)
    # create list with all IDs to check if there are duplicates with a different order
    for dup in duplicateCont[cont]:
        cont2=ExtAPI.DataModel.GetObjectById(dup)
        msgCont+=cont2.Name + ' is a duplicate of ' + cont1.Name + '\n'
        cont2.Suppress=True

if msgCont=='':
    # If message is empty, no duplicates have been found
    msg = Ansys.Mechanical.Application.Message('No duplicate contacts found', MessageSeverityType.Warning)
    ExtAPI.Application.Messages.Add(msg)
else:
```

```
msg = Ansys.Mechanical.Application.Message('Duplicates were found and suppressed: \n'+msgCont, MessageSeverity.Warning)
ExtAPI.Application.Messages.Add(msg)
```

Verify Contact Size

Goal: For a face-to-face contact, ensure that all "contact" sides are smaller than their "target" sides.

Code:

```
geom=DataModel.GeoData

contacts = DataModel.GetObjectsByType(DataModelObjectCategory.ContactRegion)

with Transaction():
    for cont in contacts:
        source_area = 0
        sourcenum = cont.SourceLocation.Ids.Count
        for x in range(0,sourcenum):
            myface = geom.GeoEntityById(cont.SourceLocation.Ids[x])
            source_area = source_area + myface.Area

        target_area = 0
        targetnum = cont.TargetLocation.Ids.Count
        for x in range(0,targetnum):
            myface = geom.GeoEntityById(cont.TargetLocation.Ids[x])
            target_area = target_area + myface.Area

        if (target_area < source_area):
            print "Flipping Source/Target For Contact Region = %s" % (cont.Name)
            cont.FlipContactTarget()

print "Done with Macro"
```

Set Pinball to 5mm for all Frictionless Contacts

Goal: Change all frictionless contacts to have a pinball of 5mm.

Code:

```
with Transaction():
    contacts = DataModel.GetObjectsByType(DataModelObjectCategory.ContactRegion)
    changeCount = 0
    for cont in contacts:
        if (cont.ContactType == ContactType.Frictionless) :
            cont.PinballRegion = ContactPinballType.Radius
            cont.PinballRadius = Quantity("5[mm]")
            changeCount = changeCount + 1
print "Done with macro, changed %s contact regions" % (changeCount)
```

Use a Named Selection as Scoping of a Load or Support

Goal: Use a named selection as scoping of a load or support object.

Code:

```
load = Model.Analyses[0].AddPressure
ns = Model.NamedSelections.Children[0]
load.Location = ns
```

Suppress Bodies Contained in a Given Named Selection

Goal: Retrieve and suppress the bodies contained in a named selection.

Code:

```
ns = Model.NamedSelections.Children[0] # selected a named selection containing bodies
bodyIds = ns.Location.Ids
with Transaction():
    for bodyId in bodyIds:
        geoBody = DataModel.GeoData.GeoEntityById(bodyId)
        body = Model.Geometry.GetBody(geoBody)
        body.Suppressed = True
print "Done with Macro"
```

Rename a Named Selection Based on Scoping

Code Purpose:

When executed, the example code shown below automatically identifies the Named Selections objects in the Outline and renames them based on the scoping specified in the Geometry Selection property. This code renames the existing Named Selections using the number of geometric entities or mesh entities included in the scoping as well as the entity type. For example: 5 Bodies, 1 Face, 75 Elements.

Example Code:

```
named_selections = DataModel.GetObjectsByType(DataModelObjectCategory.NamedSelection)

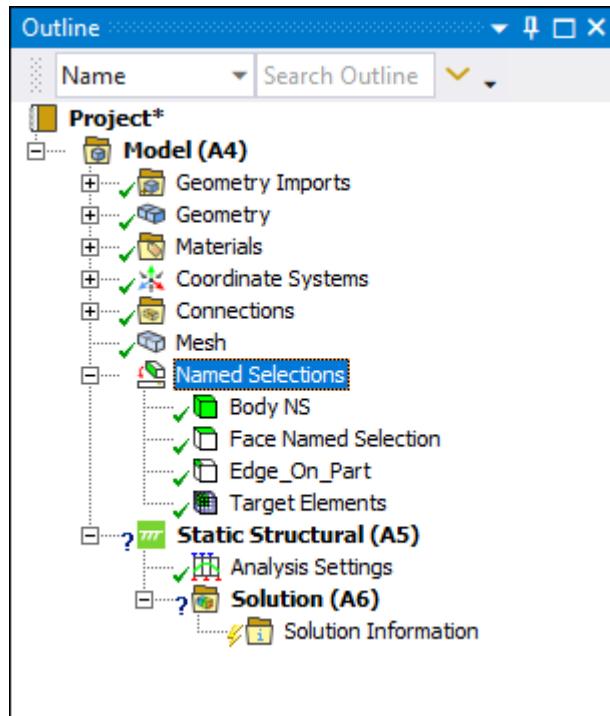
print named_selections.Count

for named_selection in named_selections:
    named_selection.Name = named_selection.PropertyByName('GeometrySelection').StringValue

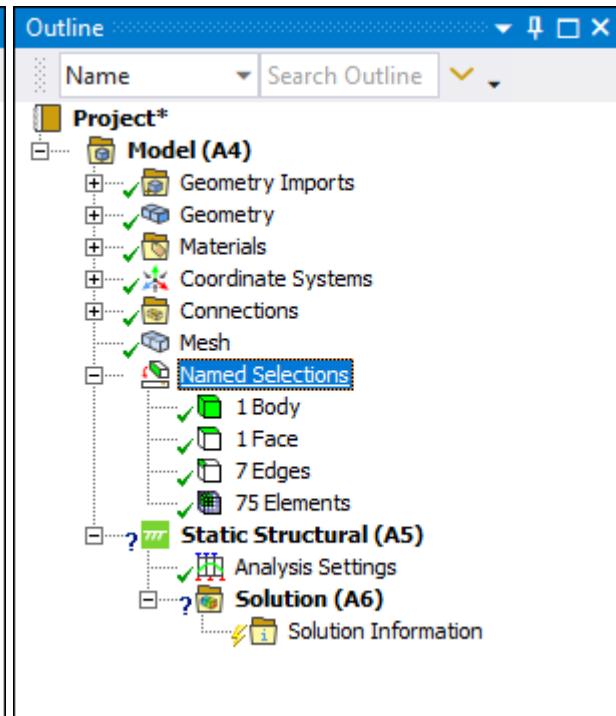
print "Done renaming!"
```

Illustrated Example

Before Script



After Script



Modify the Scoping on a Group of Objects

Goal: Loop over the selected tree objects and remove the second item to which it is scoped. This example fixes a set of pretension bolt loads that were scoped to two faces of a split cylinder and needed to be scoped to only one face.

Code:

```
with Transaction():
    for obj in Tree.ActiveObjects:
        loc = obj.Location
        loc.Ids.RemoveAt(1)
        obj.Location = loc
```

Change Tabular Data Values of a Standard Load or Support

Goal: Change the tabular data values of a standard load or support.

Code:

```
pressureLoad = Model.Analyses[0].AddPressure()
pressureLoad.Magnitude.Output.DiscreteValues = [Quantity('0 [MPa]'),Quantity('10 [MPa]')]
```

Duplicate an Harmonic Result Object

Goal: Given a selected harmonic result, duplicate it and sweep over the phase.

Code:

```
# nDiv is the number of results you want to create on a 360 basis,
# so setting to 30 will create a result every 12 degrees(360/30)
nDiv = 30
angleInc = 360/nDiv

BaseResult = Tree.FirstActiveObject

for n in range(0,nDiv+1):
    angle = Quantity(str(n*angleInc) + ' [degree]')
    newResult = BaseResult.Duplicate()
    newResult.SweepingPhase = angle
    newResult.Name = 'Sweep At ' + str(n*angleInc)
```

Retrieve Object Details Using SolverData APIs

Goal: Retrieve global and object-specific solver data from a solved analysis using **SolverData** APIs.

Code:

```
#Solver data in a solved analysis
solution = Model.Analyses[0].Solution
solver_data = solution.SolverData
solver_data.MaxElementId
solver_data.MaxNodeId
solver_data.MaxElementType
```

```
#Body data in a solved analysis
geometry = Model.Geometry
base = [i for i in geometry.GetChildren[Ansys.ACT.Automation.Mechanical.Body](True) if i.Name == 'Base'][0]
body_data = solver_data.GetObjectData(base)
body_data.ElementTypeIds
body_data.MaterialIDs
body_data.RealConstantId
```

```
#Spring data in a solved analysis
connection_group = Model.Connections
spring = [i for i in connection_group.GetChildren[Ansys.ACT.Automation.Mechanical.Connections.Spring](True) if i.N
spring_data = solver_data.GetObjectData(spring)
spring_data.RealConstantId
spring_data.ElementId
```

```
#Force data in a solved analysis
static_structural = Model.Analyses[0]
force = static_structural.Children[4]
load_data = solver_data.GetObjectData(force)
load_data.SurfaceEffectElementTypeId
```

Evaluate Spring Reaction Forces

Goal: Evaluate spring reaction forces using the **SolverData** API and the result reader.

Code:

```
# Get access to solver data
analysis = Model.Analyses[0]
solver_data = analysis.Solution.SolverData

# Get access to result reader
with analysis.GetResultsData() as reader:
    spring_results = reader.GetResult("SPRING")
    # Get a list of all springs
    springs = Model.Connections.GetChildren(DataModelObjectCategory.Spring, False)

    for spring in springs:
        print(spring.Name)

        spring_data = solver_data.GetObjectData(spring)
        element_id = spring_data.ElementId

        fForce = spring_results.GetElementValues(element_id)
        print(fForce[0])
```

Export a Result Object to an STL File

Goal: Export a result object to an STL file.

Code:

```
result = Model.Analyses[0].Solution.Children[1]
result.ExportToSTLFile("E:\\test.stl")
```

Export Result Images to Files

Goal: Export all results in the tree to PNG (2D image) and AVZ (3D image) files.

Code:

```
# get a list of all the results in the project
results = DataModel.GetObjectsByType(DataModelObjectCategory.Result)

#loop over the results
for result in results:

    # select and activate the result
    result.Activate()

    # export the result to avz file using the result name for the filename
    avzFilename = "D:\\\\Images\\\\" + result.Name + ".avz"
    Graphics.Export3D(avzFileName)

    # export the result as a 2D PNG file
    Graphics.ExportImage("D:\\\\images\\\\" + result.Name + ".png")

print "Done with Exporting Results"
```

Tag and Group Result Objects Based on Scoping and Load Steps

Goal:

- Add results based on the number of load steps
- Assign tags to created results based on scoping and type of result
- Group results based on tags

Code:

```

# Get the number of steps for the analysis
analysis_settings = Model.Analyses[0].AnalysisSettings
number_of_steps = analysis_settings.NumberOfSteps

# Get the Named Selection of interest
solution = Model.Analyses[0].Solution
bolt = DataModel.GetObjectsByName("Bolt")[0] #This is a named selection!!

with Transaction():
    # Create tags that will be used later for finding objects
    tag2 = Ansys.Mechanical.Application.ObjectTag("Bolt")
    tag3 = Ansys.Mechanical.Application.ObjectTag("U Sum")
    tag4 = Ansys.Mechanical.Application.ObjectTag("EQV")
    DataModel.ObjectTags.Add(tag2)
    DataModel.ObjectTags.Add(tag3)
    DataModel.ObjectTags.Add(tag4)

    # For each step add the desired result objects with appropriate settings
    for step in range(1, number_of_steps):
        u_result = solution.AddTotalDeformation()
        u_result.Name = "Total Deformation @ " + str(step) + " sec"
        u_result.DisplayTime = analysis_settings.GetStepEndTime(step)
        # Apply tags
        tag3.AddObject(u_result)

        s_result = solution.AddEquivalentStress()
        s_result.Name = "Eqv Stress @ " + str(step) + " sec"
        tag4.AddObject(s_result)

    for step in range(1, number_of_steps):
        u_result = solution.AddTotalDeformation()
        u_result.Name = "Bolt Deformation @ " + str(step) + " sec"
        u_result.Location = bolt
        u_result.DisplayTime = analysis_settings.GetStepEndTime(step)
        u_result.CalculateTimeHistory = False
        tag2.AddObject(u_result)
        tag3.AddObject(u_result)

        s_result = solution.AddEquivalentStress()
        s_result.Name = "Bolt Stress @ " + str(step) + " sec"
        s_result.Location = bolt
        tag2.AddObject(s_result)
        tag4.AddObject(s_result)

    tag2_list = tag2.Objects
    tag3_list = tag3.Objects
    tag4_list = tag4.Objects

    # Find similar objects using the tags
    u_all = [x for x in tag3_list if x not in tag2_list]
    u_bolt = [x for x in tag3_list if x in tag2_list]
    s_all = [x for x in tag4_list if x not in tag2_list]
    s_Bolt = [x for x in tag4_list if x in tag2_list]

    # Group similar objects
    group = Tree.Group(u_all)
    group.Name = "Total Deformation (All Bodies)"
    group = Tree.Group(u_bolt)
    group.Name = "Total Deformation (Bolt)"
    group = Tree.Group(s_all)
    group.Name = "Stress (All Bodies)"

```

```
group = Tree.Group(s_Bolt)
group.Name = "Stress (Bolt)"

Tree.Activate([solution])
```

Work with Solution Combinations

Goal: Create a solution combination object combining many environments.

Code:

```
# get the environments, an alternative would be via Model.Analyses
envs = DataModel.GetObjectsByType(DataModelObjectCategory.Analysis)

# create a solution combination object; By default it will come with 1 base case and 1 combination
sc = Model.AddSolutionCombination()

# definition object holds all the data
scdef = sc.Definition

# Any property on a base case can be set using an index and value
scdef.SetBaseCaseAnalysis(0, envs[0])
scdef.SetBaseCaseTime(0,1)

# Add more base cases as you desire
scdef.AddBaseCase()
scdef.SetBaseCaseAnalysis(1, envs[1])

# You can even pass in the Base Case settings to the constructor (Arguments are name, analysis, time)
scdef.AddBaseCase("BC 3", envs[1], 2)

# Any property on a load combination can be set using an index and value
scdef.SetLoadCombinationType(0,1)

# Add more load combinations as you desire
scdef.AddLoadCombination()
scdef.SetLoadCombinationName(1, "LC2")

# You can even pass in the Load Combination settings to the constructor (Arguments are name, type)
scdef.AddLoadCombination("LC3", 1)

# Coefficients are set using two indices and a value
scdef.SetCoefficient(0, 0, 2)
scdef.SetCoefficient(0, 2, 1)
scdef.SetCoefficient(1, 0, -0.5)
scdef.SetCoefficient(1, 1, 0.75)
scdef.SetCoefficient(2, 0, -1)
scdef.SetCoefficient(2, 2, 1.5)

#Once fully defined, add results and evaluate
sc.AddEquivalentStress()
sc.EvaluateAllResults()
```

Create a Pressure Load

Goal: Create a pressure on the first face of the first body for the first part.

Code:

```
#The following example creates a pressure on the first face of the first body for the first part.
pressure = Model.Analyses[0].AddPressure()
part = Model.Geometry.Children
```

```

body1 = part1.Children[0]
face1 = body1.GetGeoBody().Faces[0] # Get the first face of the body.
selection = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
selection.Entities = [face1]
pressure.Location = selection
pressure.Magnitude.Inputs[0].DiscreteValues = [Quantity("0 [s]"), Quantity("1 [s]")]
pressure.Magnitude.Output.DiscreteValues = [Quantity("10 [Pa]"), Quantity("20 [Pa]")]

```

Create a Convection Load

Goal

The following script creates a convection load on the first Named Selection listed in the Named Selections folder in the **Outline** using example values and Units.

Code

```

try:
    named_sel = Model.NamedSelections.Children[0]
except:
    ExtAPI.Log.WriteWarning("Named Selection not found")

convection = Model.Analyses[0].AddConvection()
if named_sel != None:
    convection.Location = named_sel

convection.AmbientTemperature.Inputs[0].DiscreteValues = [Quantity("0 [s]"), Quantity("1 [s]")] # Set the
convection.AmbientTemperature.Output.DiscreteValues = [Quantity('760 [C)'), Quantity('800 [C]')] # Set the
convection.FilmCoefficient.Inputs[0].DiscreteValues = [Quantity("0 [s]"), Quantity("1 [s]")] # Set the
convection.FilmCoefficient.Output.DiscreteValues = [Quantity("100 [W m^-1 m^-1 K^-1]"), Quantity("150 [W m^-1 m^-1 K^-1]")]

```

Create Node Merge Object at a Symmetry Plane

Goal: Create a node merge object on a model with a symmetry plane.

Code:

```

# Purpose of the script: create a node merge object on models with a (plane) symmetry
# How to use: select a coordinate that defines a symmetry plane (origin axisX, axisY)
#              then the script will create a node merge object with faces automatically selected

def GetSignedDistanceFromPointToPlane(point, planeOrigin, planeNormal):
    OP = [ planeOrigin[0]-point[0], planeOrigin[1]-point[1], planeOrigin[2]-point[2] ]
    cosOpNormal = OP[0]*planeNormal[0]+OP[1]*planeNormal[1]+OP[2]*planeNormal[2]
    normalNorm = abs(planeNormal[0]*planeNormal[0]+planeNormal[1]*planeNormal[1]+planeNormal[2]*planeNormal[2])
    dist = cosOpNormal / normalNorm
    return dist

def GetSelectedCoordinateSystem():
    if Tree.ActiveObjects.Count != 1
        return None
    obj = Tree.FirstActiveObject
    if not obj.Path.StartsWith('/Project/Model/Coordinate Systems'):
        return None
    return obj

def FindFacesClosestToPlane(planeOrigin, planeNormal, maximumDistanceToPlane):
    assembly = DataModel.GeoData.Assemblies[0]
    parts = assembly.Parts

    facesPlus = []

```

```

facesMinus = []
for part in parts:
    for body in part.Bodies:
        distBody = GetSignedDistanceFromPointToPlane(body.Centroid, planeOrigin, planeNormal)
        for face in body.Faces:
            distFace = GetSignedDistanceFromPointToPlane(face.Centroid, planeOrigin, planeNormal)
            if abs(distFace) <= maximumDistanceToPlane:
                if distBody >= 0:
                    facesPlus.append(face)
                else:
                    facesMinus.append(face)
return (facesPlus, facesMinus)

def AddNodeMergeObject():
    meshEdits = DataModel.GetObjectsByType(DataModelObjectTypeCategory.MeshEdit)
    if meshEdits.Count > 0:
        meshEdit = meshEdits[0]
    else:
        meshEdit = Model.AddMeshEdit()

    meshEdit.AddNodeMerge()
    nodeMergeObj = DataModel.GetObjectsByType(DataModelObjectTypeCategory.NodeMerge)[0]
    return nodeMergeObj

def GetGeometryBoundingBoxLength():
    geom = Model.Geometry
    lengthQuantity = geom.LengthX*geom.LengthX + geom.LengthY*geom.LengthY + geom.LengthZ*geom.LengthZ
    return lengthQuantity.Value

def ShowError(errString):
    ExtAPI.Application.ScriptByName("jscript").ExecuteCommand("WBScript.Out('" + errString + "', 1)")

def CreateNodeMergeAtPlane():
    try:
        csObj = GetSelectedCoordinateSystem()
        if csObj is None:
            raise Exception("Select a coordinate system that defines the symmetry plane as (origin, AxisX, AxisY)")

        planeOrigin = csObj.Origin
        planeNormal = csObj.ZAxis
        maximumDistanceToPlane = GetGeometryBoundingBoxLength() * 1e-5
        facesColls = FindFacesClosestToPlane(planeOrigin, planeNormal, maximumDistanceToPlane)

        nodeMergeObj = AddNodeMergeObject()

        primarySel = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
        primarySel.Entities = facesColls[0]
        secondarySel = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
        secondarySel.Entities = facesColls[1]

        nodeMergeObj.MasterLocation = primarySel
        nodeMergeObj.SlaveLocation = secondarySel
    except Exception as ex:
        ShowError("Error: {}".format(ex))

```

Access Contour Results for an Evaluated Result

Goal: For a solved result, access the nodal/elemental values and unit label.

Code:

```

Model=ExtAPI.DataModel.Project.Model
#select the result object on the tree
result=Tree.FirstActiveObject
#First result item can also be accessed with
result=Model.Analyses[0].Solution.Children[1]

```

```
#Plot the nodal/elemental values using PlotData
print("The values for " + result.Name + " is")
result.PlotData
#For accessing result, with column name ("Values")
result.PlotData["Values"]
#For accessing values on the 5th row
result.PlotData["Values"][4] #Array starts at 0
#For accessing result value components
result.PlotData.Dependents
#For accessing node/element IDs
result.PlotData.Independents
#To get the unit label for the values
result.PlotData["Values"].Unit
```

Write Contour Results to a Text File

Goal: For a solved result, write the nodal/elemental values to a text file using **PlotData**.

Code:

```
Model=ExtAPI.DataModel.Project.Model
#select the result object on the tree
result=Tree.FirstActiveObject
resultValues= result.PlotData["Values"]
nodeList=result.PlotData["Node"]
with open('C:\Users\Admin\Desktop\Testfile.txt','w') as testfile:
    for ii in range(len(nodeList)):
        a=nodeList[ii]
        b=resultValues[ii]
        wrt=str(ii)+'\t'+str(a)+"\t"+str(b)+"\n"
        testfile.write(wrt)
```

Access Contour Results at Individual Nodes/Elements

Goal: For a solved result, get the result value at an individual node using **PlotData**.

Code:

```
Model=ExtAPI.DataModel.Project.Model

#select the result object on the tree
result=Tree.FirstActiveObject
plotDataResult= result.PlotData

#For node number = nodeID
def findNodeResult(nodeID,plotDataResult):
    nodes=plotDataResult ['Node']
    resultValue=plotDataResult ['Values']
    if nodeID in nodes:

        for index in range(len(nodes)):

            if nodes[index]==nodeID:
                return resultValue[index]
    else:
        print("Given NodeID doesn't exist in result set")

#Find the result value associated for node number 3 using
findNodeResult(3,plotDataResult)
```

Set Arbitrary Coordinate System Properties

The script illustrated below enables you to:

1. Create a new Coordinate System.
2. Specify values for an arbitrary origin. This specifies the Details properties **Origin X**, **Origin Y**, and **Origin Z**.
3. Set an arbitrary direction vector. This specifies the Details properties **X Axis Data**, **Y Axis Data**, and **Z Axis Data**.

Important:

The ability to set arbitrary direction vectors is only available through scripting.

Example Code

```
# Create a new coordinate system
csys = Model.CoordinateSystems.AddCoordinateSystem()

# place csys origin at arbitrary (0,25,50) location
csys.SetOriginLocation(Quantity(0,"in"), Quantity(25,"in"), Quantity(50,"in"))

# set primary X axis to arbitrary (1,2,3) direction
csys.PrimaryAxisDirection = Vector3D(1,2,3)

# force a graphics redraw to update coordinate system graphics annotations
csys.Suppressed=True
csys.Suppressed=False
```

Transform Coordinate Systems (with Math)

Goal: Transform the given global coordinates to the first non-global coordinate system object in the tree.

Code:

```
import units
import math

def MatrixTransformationGlobalToUserCS(global_coordinates, destination_coordinate_system, length_unit):
    from_unit = DataModel.CurrentConsistentUnitFromQuantityName("Length")
    factor = units.ConvertUnit(1, from_unit, length_unit, "Length")

    origin = destination_coordinate_system.Origin

    x_axis = destination_coordinate_system.XAxis
    y_axis = destination_coordinate_system.YAxis
    z_axis = destination_coordinate_system.ZAxis

    user_coordinates = []
    user_coordinates.append(x_axis[0]*(global_coordinates[0] - factor*origin[0]) +
                           x_axis[1]*(global_coordinates[1] - factor*origin[1]) +
                           x_axis[2]*(global_coordinates[2] - factor*origin[2]))
    user_coordinates.append(y_axis[0]*(global_coordinates[0] - factor*origin[0]) +
                           y_axis[1]*(global_coordinates[1] - factor*origin[1]) +
                           y_axis[2]*(global_coordinates[2] - factor*origin[2]))
```

```

user_coordinates.append(z_axis[0]*(global_coordinates[0] - factor*origin[0]) +
                        z_axis[1]*(global_coordinates[1] - factor*origin[1]) +
                        z_axis[2]*(global_coordinates[2] - factor*origin[2]))

if destination_coordinate_system.CoordinateSystemType == CoordinateSystemTypeEnum.Cartesian:
    return user_coordinates
elif destination_coordinate_system.CoordinateSystemType == CoordinateSystemTypeEnum.Cylindrical:
    r = sqrt(user_coordinates[0] * user_coordinates[0] + user_coordinates[1] * user_coordinates[1])
    theta = math.degrees(math.atan(user_coordinates[1] / user_coordinates[0]))
    z = user_coordinates[2]
    return [r, theta, z]

def MatrixTransformationGlobalToUserCSUsingMatrix4D(global_coordinates, destination_coordinate_system, length_unit):
    from_unit = ExtAPI.DataModel.CurrentConsistentUnitFromQuantityName("Length")
    factor = units.ConvertUnit(1, from_unit, length_unit, "Length")

    origin = destination_coordinate_system.Origin

    x_axis = destination_coordinate_system.XAxis
    y_axis = destination_coordinate_system.YAxis
    z_axis = destination_coordinate_system.ZAxis

    x_axis_vector = Vector3D(x_axis[0], x_axis[1], x_axis[2])
    y_axis_vector = Vector3D(y_axis[0], y_axis[1], y_axis[2])
    z_axis_vector = Vector3D(z_axis[0], z_axis[1], z_axis[2])

    identity = Matrix4D()
    transformation = identity.CreateSystem(x_axis_vector, y_axis_vector, z_axis_vector)

    vector = Vector3D(global_coordinates[0] - origin[0] * factor, global_coordinates[1] - origin[1] * factor, global_coordinates[2] - origin[2] * factor)
    transformation.Transpose()
    vector_trans = transformation.Transform(vector)
    return vector_trans

global_coordinates = [20, 40, 60]
localCS = Model.CoordinateSystems.Children[1]

localCoordinates1 = MatrixTransformationGlobalToUserCS(global_coordinates, localCS, "mm")
localCoordinates2 = MatrixTransformationGlobalToUserCSUsingMatrix4D(global_coordinates, localCS, "mm")

print(localCoordinates1)
print(localCoordinates2)

```

Select Objects By Name

Goal

The following script examples enable you to select an object based on its name.

Code

```

# Goal: Get the first body whose name contains the specified number
def GetBodyByPartNumber(number):
    bodies = DataModel.Project.Model.Geometry.GetChildren(DataModelObjectCategory.Body, True)
    for body in bodies:
        if str(number) in body.Name:
            return body

    ExtAPI.Log.WriteWarning("No bodies were found containing " + str(number))
    return None

# Goal: Get objects whose name contains the specified string
def GetObjectsByPartialName(name):
    object_list = []
    child_list = Tree.AllObjects

```

```
for child in child_list:
    if name in child.Name:
        object_list.Add(child)

return object_list

# Goal: Get the first Named Selection whose name contains the specified string
def GetNamedSelectionByName(name,AcceptPartial = False):
    try:
        NSS = ExtAPI.DataModel.Project.Model.NamedSelections.Children
    except:
        return None
    for ns in NSS:
        if ns.Name.ToLower() == name.ToLower():
            return ns
        if AcceptPartial and (name.ToLower() in ns.Name.ToLower()):
            return ns
    ExtAPI.Log.WriteWarning("Named Selection not found: "+str(name))
    return None

# Goal: Get Named Selections whose name contains the specified string
def GetNamedSelectionsByName(name,AcceptPartial = False):
    try:
        NSS = ExtAPI.DataModel.Project.Model.NamedSelections.Children
    except:
        return []
    if AcceptPartial:
        return [ns for ns in NSS if name.ToLower() in ns.Name.ToLower()]
    else:
        return [ns for ns in NSS if ns.Name.ToLower() == name.ToLower()]
```

Export Figures

Goal

The following script enables you to automatically export any [Figures](#) included in your analysis to a specified folder. Figures are often useful for examining results.

Code

```
# Export to jpeg files a selection of figures from the tree
# Pictures are stored in the corresponding analysis folder
# Created by P. Thieffry
# Last update: 2021/04/14

clr.AddReference("Ans.UI.Toolkit.Base")
clr.AddReference("Ans.UI.Toolkit")
from Ansys.UI.Toolkit import *

figures=ExtAPI.DataModel.Tree.ActiveObjects

if (figures.Count == 0):
    mess_line1 = 'Please select at least one figure'
    MessageBox.Show(mess_line1)
    pass

if (figures[0].GetType()!= Ansys.ACT.Automation.Mechanical.Figure):
    mess_line1 = 'Please select figure(s)'
    MessageBox.Show(mess_line1)
    pass

import wbjn
cmd = 'returnValue(GetUserFilesDirectory())'
user_dir = wbjn.ExecuteCommand(ExtAPI,cmd)

for fig in figures:
```

```
if fig.GetType() != Ansys.ACT.Automation.Mechanical.Figure:  
    continue  
  
ExtAPI.DataModel.Tree.Activate(fig)  
anWD=ExtAPI.DataModel.AnalysisById(0).WorkingDir  
filename=user_dir+'\\'+Figure - '+fig.Parent.Name+'.jpg'  
ExtAPI.Graphics.ExportImage(filename)  
msg = Ansys.Mechanical.Application.Message('Figure for '+fig.Parent.Name+' stored in folder '+user_dir, Me  
ExtAPI.Application.Messages.Add(msg)
```

Script Examples for Interacting with the Mechanical Session

The following scripts are for interacting with the Mechanical session:

- Remesh a Model Multiple Times and Track Metrics
- Perform Solution While Specifying Solution Handler and the Number of Cores
- Scan Results, Suppress Any with Invalid Display Times, and Evaluate
- Check Version
- Check Operating Environment
- Search for Keyword and Export
- Modify Export Setting
- Pan the Camera
- Functions to Draw
- Export All Result Animations
- Get User Files Directory
- Display an Arrow at the Centroid of Selected Faces
- Compute Shortest Distance Between Two Faces
- Display Extensions Loaded in Mechanical

Remesh a Model Multiple Times and Track Metrics

Goal: Remesh a model five times, tracking how long each remesh takes and the number of nodes that are created.

Code:

```
from time import clock, time
for x in range(0, 5):
    Model.ClearGeneratedData()
    t1 = time()
    Model.Mesh.GenerateMesh()
    t2 = time()
    print "Stats for mesh %d, elapsed time=%d" % (x+1, t2-t1)
    print Model.Mesh.Nodes
```

Perform Solution While Specifying Solution Handler and the Number of Cores

Goal

The following script executes the solution process and:

1. Sets the Solution Process Settings to the **My Computer, Background** setting.
2. Completes solutions for core settings.
3. Tracks how long each solution takes when solved using a different number of cores.

Code

```
# Get the existing My Computer, Background solve configuration
config = ExtAPI.Application.SolveConfigurations["My Computer, Background"]
config.SetAsDefault()

for cores in range(8,13):
    config.SolveProcessSettings.MaxNumberOfCores = cores
    Model.Analyses[0].Solution.ClearGeneratedData()
    # pass in True to wait for the solve to complete
    Model.Analyses[0].Solution.Solve(True)
    Model.Analyses[0].Solution.GetResults()
    elapsed_time = Model.Analyses[0].Solution.ElapsedRunTime
    print "%d cores took %d secs" % (cores, elapsed_time)
```

Scan Results, Suppress Any with Invalid Display Times, and Evaluate

Goal: Scan all result objects in your first analysis, suppress any results with invalid display times, and then evaluate all results.

Code:

```
aset = Model.Analyses[0].AnalysisSettings
OrigStep = aset.CurrentStepNumber
aset.CurrentStepNumber = aset.NumberOfSteps
FinalTime = aset.StepEndTime
aset.CurrentStepNumber = OrigStep
sol = Model.Analyses[0].Solution
for obj in sol.Children:
    if hasattr(obj,"DisplayTime"):
        if obj.DisplayTime > FinalTime:
            obj.Suppressed = True
sol.EvaluateAllResults()
```

Check Version

Goal: Check the version in which your user is operating.

Code:

```
clr.AddReference("Ans.Utilities")
import Ansys.Utilities
from Ansys.Utilities import ApplicationConfiguration
```

```
---  
ansysVersion = ApplicationConfiguration.DefaultConfiguration.VersionInfo.VersionString
```

Check Operating Environment

Goal: Check to see if your user is operating in a Unix environment.

Code:

```
clr.AddReference("Ans.Utilities")
import Ansys.Utilities
from Ansys.Utilities import ApplicationConfiguration
---
isUnix = ApplicationConfiguration.DefaultConfiguration.IsUnix
```

Search for Keyword and Export

Goal: Loop through all results that contain a keyword of **For Image** and export the image shown to a directory.

You must specify the path in the first line.

Code:

```
path = "C:\\\\"
n = 0

for analysis in Model.Analyses:
    sol = analysis.Solution
    for sol_obj in sol.Children:
        if sol_obj.Name.Contains("For_Image"):
            n += 1
            sol_obj.Activate()
            Graphics.ExportImage(path + sol_obj.name + ".png")
```

Modify Export Setting

Goal: Modify the export setting to include node numbers in exports.

Code:

```
ExtAPI.Application.ScriptByName("jscript").ExecuteCommand('WB.PreferenceMgr.Preference("PID_Show_Node_Numbers") =
```

Pan the Camera

Goal: Pan the camera.

Code:

```
camera = Graphics.Camera
up_vector = camera.UpVector
view_vector = camera.ViewVector
```

```
#get the 2D CSYS in the screen plane based on the computed UpVector, derived from the prescribed one.
plane_right = up_vector.CrossProduct(view_vector) # the "x" axis of the 2D CSYS.
plane_up = up_vector - (up_vector.DotProduct(view_vector)) * view_vector # the "y" axis of the 2D CSYS.

#construct the pan vector in the screen plane (Use the units from Graphics.Unit).
pan_right = 100
pan_up = 100
pan_vector = plane_right * pan_right + plane_up * pan_up

#set the new focal point by adding the pan vector to the original focal point.
pan_origin = camera.FocalPoint.Location
new_x, new_y, new_z = (pan_origin[0] + pan_vector[0], pan_origin[1] + pan_vector[1], pan_origin[2] + pan_vector[2])
camera.FocalPoint = Point([new_x, new_y, new_z], Graphics.Unit)
```

Functions to Draw

Goal: Add several example draw functions that can be used to draw entities in the Mechanical graphics window.

Code:

```
def DrawElements():
    elems = DataModel.MeshDataByName('Global').Elements
    elem = Graphics.Scene.Factory3D.CreateMesh(list(elems)[:5])
    elem.Color = 0xA00ABC

def DrawBody():
    body = DataModel.GeoData.Assemblies[0].Parts[0].Bodies[0]
    geo = Graphics.Scene.Factory3D.CreateGeometry(body)
    geo.Color = 0xAAA000

def DrawFace():
    face = DataModel.GeoData.Assemblies[0].Parts[0].Bodies[0].Faces[0]
    geo = Graphics.Scene.Factory3D.CreateGeometry(face)
    geo.Color = 0xABCA0C

def DrawEdge():
    edge = DataModel.GeoData.Assemblies[0].Parts[0].Bodies[0].Faces[0].Edges[1]
    geo = Graphics.Scene.Factory3D.CreateGeometry(edge)
    geo.LineWeight = 14
    geo.VertexColor = 0x0000AA
    geo.VertexSize = 15
    geo.Color = 0x000ABC

def DrawVertex():
    vertex = DataModel.GeoData.Assemblies[0].Parts[0].Bodies[0].Faces[0].Edges[0].Vertices[0]
    geo = Graphics.Scene.Factory3D.CreateGeometry(vertex)
    geo.Color = 0xBACBAC

def DrawTriad():
    triad = Graphics.Scene.Factory3D.CreateTriad(1.0)

def DrawArrow():
    Vector3D = Graphics.CreateVector3D
    arrow = Graphics.Scene.Factory3D.CreateArrow(0.5)
    arrow.Color = 0xFFABC0
    arrow.Transformation3D.Translate(Vector3D(0.5, 0.5, 1))

def DrawBox():
    box = Graphics.Scene.Factory3D.CreateBox(2.0, 3.0, 4.0)
    box.Color = 0xFF0ABC

def DrawCircle():
    cone = Graphics.Scene.Factory3D.CreateCircle(0.5)
    cone.Color = 0xAAAA00
```

```

def DrawCone():
    cone = Graphics.Scene.Factory3D.CreateCone(3.0, 2.0, 1.0)
    cone.Color = 0xABC000

def DrawCylinder():
    cone = Graphics.Scene.Factory3D.CreateCylinder(0.5, 2)
    cone.Color = 0xAAA000

def DrawDisc():
    cone = Graphics.Scene.Factory3D.CreateDisc(0.5)
    cone.Color = 0xAAABBB

def DrawQuad():
    sphere = Graphics.Scene.Factory3D.CreateQuad(0.33, 0.33)
    sphere.Color = 0xFFA000

def DrawShell():
    shell = Graphics.Scene.Factory3D.CreateShell([1., 1., 1., 2., 1., 1., 1., 1., 1., 2.], [0., 1., 0., 0., 1., 0., 0., 1., 0., 0.])
    shell.Color = 0xFFFFFFF

def DrawSphere():
    sphere = Graphics.Scene.Factory3D.CreateSphere(0.33)
    sphere.Color = 0xFF0A00

def DrawPolyline3D():
    Point2D = Graphics.CreatePixelPoint
    Point3D = Graphics.CreateWorldPoint
    Vector3D = Graphics.CreateVector3D

    with Graphics.Suspend():
        p1 = Point2D(10, 10)
        p2 = Point2D(10, 100)
        p3 = Point2D(100, 100)
        p4 = Point2D(100, 10)

        coll = Graphics.Scene.CreateChildCollection()
        l1 = coll.Factory2D.CreatePolyline([p1, p2, p3, p4])
        l1.Closed = True

        p5 = Point2D(0, 0)
        p6 = Point2D(100, 100)
        l2 = Graphics.Scene.Factory2D.CreatePolyline([p5, p6])

        p7 = Point2D(20, 40)
        text = Graphics.Scene.Factory2D.CreateText(p7, "Hello World 3D")

        wp1 = Point3D(0, 5, 0)
        wp2 = Point3D(0, 0, 0)
        wp3 = Point3D(5, 0, 0)

        coll = Graphics.Scene.CreateChildCollection()

        l1 = coll.Factory3D.CreatePolyline([wp1, wp2])
        l2 = coll.Factory3D.CreatePolyline([wp2, wp3])

#shell = Graphics.Scene.Factory3D.CreateShell([1.,1.,1.,2.,1.,1.,1.,1.,2.], [0.,1.,0.,0.,1.,0.,0.,1.,0.], [0,1,0,1,0,1,0,1,0])

points = []
for i in range(10):
    for j in range(10):
        for k in range(10):
            points.Add(Point3D(float(i)/float(10), float(j)/float(10), float(k)/float(10)))

coll.Factory3D.CreatePoint(points, 4)

return True

for i in range(5):
    for j in range(5):
        for k in range(5):
            point = Point3D(float(i)/float(2), float(j)/float(2), float(k)/float(2))

```

```

        coll.Factory2D.CreateText(point, str(i + j + k))

def DrawPolyline2D():
    Point2D = Graphics.CreatePixelPoint
    Point3D = Graphics.CreateWorldPoint

    with Graphics.Suspend():

        p1 = Point2D(10, 10)
        p2 = Point2D(10, 100)
        p3 = Point2D(100, 100)
        p4 = Point2D(100, 10)

        coll = Graphics.Scene.CreateChildCollection()
        l1 = coll.Factory2D.CreatePolyline([p1, p2, p3, p4])
        l1.Closed = True

        p5 = Point2D(0,0)
        p6 = Point2D(100,100)
        l2 = Graphics.Scene.Factory2D.CreatePolyline([p5, p6])

        p7 = Point2D(20,40)
        text = Graphics.Scene.Factory2D.CreateText(p7, "Hello World 2D")

        wp1 = Point3D(0,5,0)
        wp2 = Point3D(0,0,0)
        wp3 = Point3D(5,0,0)

        coll = Graphics.Scene.CreateChildCollection()

        l1 = coll.Factory3D.CreatePolyline([wp1, wp2])
        l2 = coll.Factory3D.CreatePolyline([wp2, wp3])

        points = []
        for i in range(10):
            for j in range(10):
                points.Add(Point2D(float(i)/float(10), float(j)/float(10)))

        coll.Factory2D.CreatePoint(points, 4)

def DrawText2D(pixX,pixY,color):
    Point2D = Graphics.CreatePixelPoint
    with Graphics.Suspend():
        p7 = Point2D(pixX,pixY)
        text = Graphics.Scene.Factory2D.CreateText(p7, "Hello World 2D")
        text.Color = color

def DrawText3D(x,y,z,color):
    Point3D = Graphics.CreateWorldPoint
    with Graphics.Suspend():
        p7 = Point3D(x,y,z)
        text = Graphics.Scene.Factory3D.CreateText3D(p7, "Hello World 3D")
        text.Color = color

```

Export All Result Animations

Goal: Export the animation of the results in the model to a WMV video file with given resolution, frames and duration.

Code:

```

#Set camera and view
cam = Graphics.Camera
cam.SetSpecificViewOrientation(ViewOrientationType.Right)

#Animation settings

```

```

Graphics.ResultAnimationOptions.NumberOfFrames = 10
Graphics.ResultAnimationOptions.Duration = Quantity(2, 's')
settings = Ansys.Mechanical.Graphics.AnimationExportSettings(width = 1000, height = 665)

for analysis in Model.Analyses:
    results = analysis.Solution.GetChildren(DataModelObjectCategory.Result, True)
    for result in results:
        location ="C:\\Samples\\WMV\\\" + result.Name + ".wmv"
        result.ExportAnimation(location, GraphicsAnimationExportFormat.WMV, settings)

```

Get User Files Directory

Goal

The following script queries the Workbench application for the directory path for user files.

Code

```

import wbjn
cmd = 'returnValue(GetUserFilesDirectory())'
user_dir = wbjn.ExecuteCommand(ExtAPI,cmd)
print user_dir

```

Display an Arrow at the Centroid of Selected Faces

Goal

The following script creates an arrow annotation and places it at the centroid on each specified face. The length of each arrow is half of the value of the **Bounding Box** of the face.

Code

```

for geoId in ExtAPI.SelectionManager.CurrentSelection.Ids :
    geoEntity=ExtAPI.DataModel.GeoData.GeoEntityById(geoId)
    if geoEntity.Type != GeoCellTypeEnum.GeoFace :
        print("Only faces are supported")
        continue

    # Get the centroid of the face in CAD units
    locationCAD = geoEntity.Centroid

    # Get the normal of the face at the centroid
    params = geoEntity.ParamAtPoint(geoEntity.Centroid)
    direction = geoEntity.NormalAtParam(params[0], params[1])

    # Create the arrow, set size at half the bounding box length of the face
    bbox = geoEntity.GetBoundingBox()
    sizeCAD = sqrt((bbox[0]-bbox[3])**2 + (bbox[1]-bbox[4])**2 + (bbox[2]-bbox[5])**2)
    arrow = ExtAPI.Graphics.Scene.Factory3D.CreateArrow(sizeCAD/2)
    arrow.Visible = False
    # Set the color to red
    arrow.Color=0xFF0000

    # Align the arrow
    x = direction[0]
    y = direction[1]
    z = direction[2]
    r = sqrt(y*y+z*z)
    arrow.Transformation3D.Rotate(ExtAPI.Graphics.CreateVector3D(0,1,0), atan2(x,r))
    arrow.Transformation3D.Rotate(ExtAPI.Graphics.CreateVector3D(1,0,0), atan2(z,y) - pi/2.0)
    arrow.Transformation3D.Translate(locationCAD[0],locationCAD[1],locationCAD[2])

```

```

ExtAPI.Graphics.Scene.Visible = True
arrow.Visible = True
print user_dir

```

Compute Shortest Distance Between Two Faces

Goal

The following script enables you to calculate the shortest distance between two faces.

Code

```

# Compute distance between two faces
import math
import units
clr.AddReference("Ans.UI.Toolkit.Base")
clr.AddReference("Ans.UI.Toolkit")
from Ansys.UI.Toolkit import *

# Utility functions
def distance(p1,p2):
    return math.sqrt((p2[0]-p1[0])**2+(p2[1]-p1[1])**2+(p2[2]-p1[2])**2)

def dotProduct(v1,v2):
    return v2[0]*v1[0]+v2[1]*v1[1]+v2[2]*v1[2]

def vecLength(vec):
    return math.sqrt(vec[0]**2+vec[1]**2+vec[2]**2)

class BoundingBoxData:
    # Class to compute geometric elements of a bounding box such as elementary vectors, edge length and midpoint
    # Assumes ABCDEFGH points where A is origin, G is diagonal point, orientation triad is AB / AC /AE
    def __init__(self,face):
        bbox=face.GetBoundingBox()
        self.Face=face
        self.Origin=[bbox[0],bbox[1],bbox[2]]
        self.Diagonal=[bbox[3],bbox[4],bbox[5]]
        self.Midpoint=[0.5*(self.Diagonal[0]+self.Origin[0]),0.5*(self.Diagonal[1]+self.Origin[1]),0.5*(self.Diagonal[2]+self.Origin[2])]
        pointB=[bbox[3],bbox[1],bbox[2]]
        pointC=[bbox[0],bbox[4],bbox[2]]
        pointE=[bbox[0],bbox[1],bbox[5]]

        self.vecX=[pointB[0]-self.Origin[0],pointB[1]-self.Origin[1],pointB[2]-self.Origin[2]]
        self.vecY=[pointC[0]-self.Origin[0],pointC[1]-self.Origin[1],pointC[2]-self.Origin[2]]
        self.vecZ=[pointE[0]-self.Origin[0],pointE[1]-self.Origin[1],pointE[2]-self.Origin[2]]
        self.xlength=distance(pointB,self.Origin)
        self.ylength=distance(pointC,self.Origin)
        self.zlength=distance(pointE,self.Origin)

        for i in range(0,3):
            if self.xlength != 0.: self.vecX[i]=self.vecX[i]/self.xlength
            if self.ylength != 0.: self.vecY[i]=self.vecY[i]/self.ylength
            if self.zlength != 0.: self.vecZ[i]=self.vecZ[i]/self.zlength

    def CheckPointInBox(self,point):
        # Check if a point [x,y,z] lies within the bounding box
        isInside=True
        vector=[point[0]-self.Midpoint[0],point[1]-self.Midpoint[1],point[2]-self.Midpoint[2]]
        if self.xlength != 0:
            checkX=abs(dotProduct(vector,self.vecX))-self.xlength/2<=1e-8
        else:
            checkX=abs(point[0]-self.Origin[0])<1e-8

        if self.ylength != 0:
            checkY=abs(dotProduct(vector,self.vecY))-self.ylength/2<=1e-8

```

```

    else:
        checkY=abs(point[1]-self.Origin[1])<1e-8

    if self.zlength != 0:
        checkZ=abs(dotProduct(vector,self.vecZ))-self.zlength/2<=1e-8
    else:
        checkZ=abs(point[2]-self.Origin[2])<1e-8

    if not(checkX and checkY and checkZ):
        isInside=False

    return isInside

def checkEdgeDistance(edge,bbox):
    # Compute distance between an edge and a face
    # edge and face from bbox are the ones to compare
    # bbox is the boundingboxdata object related to face
    # start_vertex holds the coordinates of the end vertex of the edge to start from when scanning the edge (in [x,y,z])
    #
    mindist=1e31

    deltaP=0.05 # increment in parametric value on edge
    p0=0

    face=bbox.Face

    while p0<=1:
        vert=edge.PointAtParam(p0) # get point at p0
        pa=face.ParamAtPoint((vert[0],vert[1],vert[2])) # project on face
        pt=face.PointAtParam(pa[0],pa[1]) # from (u,v), compute projection coordinates
        if bbox.CheckPointInBox(pt): # verify if projection is on the face or not using bounding box
            dist=distance(vert,pt)
            if dist<mindist:
                mindist=dist
        p0=p0+deltaP
    return mindist

minDistSource={'f1vf2': 'between first face vertices and second face',
               'f2vf1': 'between second face vertices and first face',
               'f1vf2e': 'between first face vertices and second face edges',
               'f2vf1e': 'between second face vertices and first face edges',
               'f1vf2mod': 'between first face edges and second face',
               'f2vf1mod': 'between second face edges and first face',
               }

curSel=ExtAPI.SelectionManager.CurrentSelection

# Need to have two faces selected, otherwise pass
mess_line1 = 'Please select 2 faces'
if (curSel.Ids.Count!=2):
    MessageBox.Show(mess_line1)
    pass

face=ExtAPI.DataModel.GeoData.GeoEntityById(curSel.Ids[0])
if face.Type != GeoCellTypeEnum.GeoFace:
    MessageBox.Show(mess_line1)
    pass

# Check if selection contains faces, otherwise don't do anything
f1=ExtAPI.DataModel.GeoData.GeoEntityById(curSel.Ids[0])
f2=ExtAPI.DataModel.GeoData.GeoEntityById(curSel.Ids[1])

# Compute bounding box data once to be used in later checks
bbox1=BoundingBoxData(f1)
bbox2=BoundingBoxData(f2)

edgef1=f1.Edges
edgef2=f2.Edges

edgef1Id=[]

```

```

for edge in edgef1:
    edgef1Id.append(edge.Id)

edgef2Id=[]
for edge in edgef2:
    edgef2Id.append(edge.Id)

# Collect vertices from the two faces - if no vertices, retrieve points from edges
vertf1=[]
if len(f1.Vertices)!=0:
    for vert in f1.Vertices:
        vertf1.append([vert.X,vert.Y,vert.Z])
else:
    for edge in edgef1:
        for i in range(0,len(edge.Points)/3):
            vertf1.append([edge.Points[3*i],edge.Points[3*i+1],edge.Points[3*i+2]])

vertf2=[]
if len(f2.Vertices)!=0:
    for vert in f2.Vertices:
        vertf2.append([vert.X,vert.Y,vert.Z])
else:
    for edge in edgef2:
        for i in range(0,len(edge.Points)/3):
            vertf2.append([edge.Points[3*i],edge.Points[3*i+1],edge.Points[3*i+2]])

# Compute shortest distance
mindist=1e31
minSource=''

# Run 4 trials:
# - f1 vertices projected on f2
# - f2 vertices projected on f1
# - f1 vertices projected on edges of f2
# - f2 vertices projected on edges of f1

for vert in vertf1: # Compare vertices in f1 vs projection on f2
    pa=f2.ParamAtPoint((vert[0],vert[1],vert[2])) # project vert on f2
    pt=f2.PointAtParam(pa[0],pa[1]) # from (u,v), compute projection coordinates
    if bbox2.CheckPointInBox(pt): # verify if projection is on the face or not using bounding box
        dist=distance(vert,pt)
        if dist < mindist:
            mindist=dist
            minVertex=vert
            minSource='flvf2'

if mindist!=1e31: # a minimum has been found, scan edges related to closest vertex to see if a closer location
    for vert in f1.Vertices:
        if vert.X==minVertex[0] and vert.Y==minVertex[1] and vert.Z==minVertex[2]:
            for edge in vert.Edges: # scan edges related to closest vertex
                if edge.Id in edgef1Id: # scan only edges related to face 1
                    mindist2=checkEdgeDistance(edge,bbox2)
                    if mindist2 < mindist:
                        mindist=mindist2
                        minSource='flvf2mod'

for vert in vertf2: # Compare vertices in f2 vs projection on f1 - same as above
    pa=f1.ParamAtPoint((vert[0],vert[1],vert[2])) # from (u,v), compute projection coordinates
    pt=f1.PointAtParam(pa[0],pa[1])
    if bbox1.CheckPointInBox(pt):
        dist=distance(vert,pt)
        if dist < mindist:
            mindist=dist
            minVertex=vert
            minSource='f2vf1'

if minSource=='f2vf1': # a minimum has been found, scan edges related to closest vertex to see if a closer location
    for vert in f2.Vertices:
        if vert.X==minVertex[0] and vert.Y==minVertex[1] and vert.Z==minVertex[2]:
            for edge in vert.Edges:
                if edge.Id in edgef2Id:

```

```

        mindist2=checkEdgeDistance(edge,bbox1)
        if mindist2 < mindist:
            mindist=mindist2
            minSource='f2vf1mod'

for vert in vertf1: # Compare vertices in f1 vs projection on edges of f2
    for edge in edgef2:
        pa=edge.ParamAtPoint((vert[0],vert[1],vert[2]))
        if pa>=0 and pa<=1: # verify if projection is on the edge using reduced coordinate
            pt=edge.PointAtParam(pa)
            dist=distance(vert,pt)
            if dist < mindist:
                mindist=dist
                minSource='f1vf2e'

for vert in vertf2: # Compare vertices in f2 vs projection on edges of f1
    for edge in edgef1:
        pa=edge.ParamAtPoint((vert[0],vert[1],vert[2]))
        if pa>=0 and pa<=1:
            pt=edge.PointAtParam(pa)
            dist=distance(vert,pt)
            if dist < mindist:
                mindist=dist
                minSource='f2vf1e'

length_unit=ExtAPI.DataModel.GeoData.Unit
curUnit=ExtAPI.DataModel.CurrentUnitFromQuantityName("Length")

# Convert to Mechanical units
mindist_curUnit=units.ConvertUnit(mindist, fromUnit=length_unit, toUnit=curUnit)

# Display result
mess_line = 'Computed distance (approx) = ' + str(round(mindist_curUnit,3)) + ' ' + str(curUnit) + '\n'
#mess_line += '(found '+minDistSource[minSource]+')\n'
MessageBox.Show(mess_line)

```

Display Extensions Loaded in Mechanical

Code Purpose:

This script displays a message box that lists all of extensions that you have loaded into Mechanical.

Example Code:

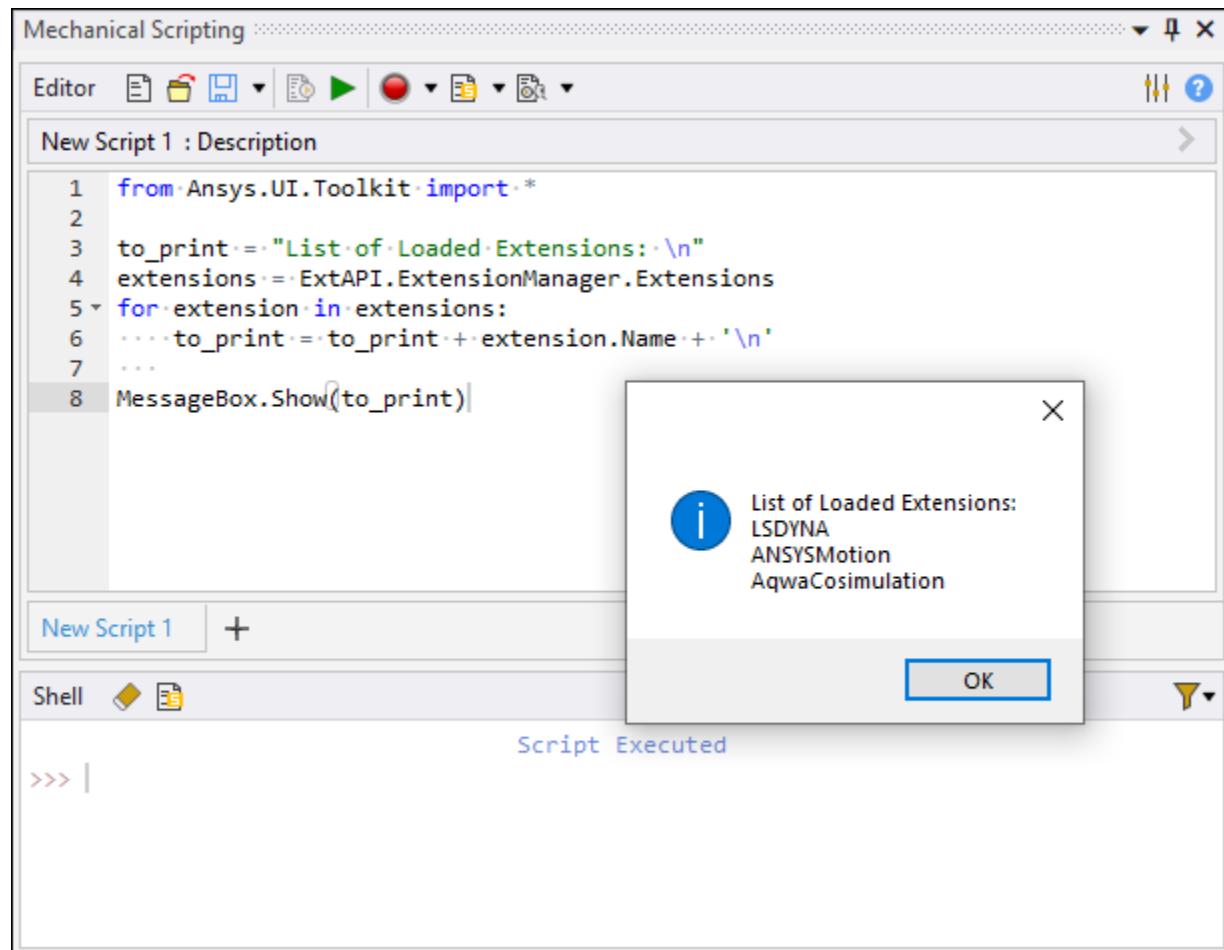
```

to_print = "List of Loaded Extensions: \n"
extensions = ExtAPI.ExtensionManager.Extensions
for extension in extensions:
    to_print = to_print + extension.Name + '\n'

MessageBox.Show(to_print)

```

Illustrated Example



Script Examples for Interacting with Results

The following scripts are for generating results in the Mechanical application:

[Retrieve Stress Results](#)

[Create Probe Principal Stresses from a Node Selection](#)

[Finding Hot Spots](#)

Retrieve Stress Results

Goal: Obtain stress results for an element.

Code:

```
reader = Model.Analyses[0].GetResultsData()
reader.CurrentResultSet=1
S=reader.GetResult("S")
S.GetElementValues(1)
reader.Dispose()
```

Important:

Principal stresses obtained from the above script could differ from the same value calculated in Mechanical.

Note that the command `reader.GetResult("PRIN_S")` returns the principal stress values on a per element basis and then averages these values from the elements at a common node. (Similar to AVPRIN,1 in Mechanical APDL)

In Mechanical, the component values are first averaged from the elements at a common node, and then the principal stress values are calculated from the averaged components. (Similar to **AVPRIN,0** in Mechanical APDL).

Create Probe Principal Stresses from a Node Selection

Goal

The following script inserts a Probe for Principal Stresses and specifies a node for solution.

Code

```
# Retrieve principal stresses from a selection of nodes and display in a message box

import math
import units
import mech_dpf
import Ans.DataProcessing as dpf
```

```
clr.AddReference("Ans.UI.Toolkit.Base")
clr.AddReference("Ans.UI.Toolkit")
from Ansys.UI.Toolkit import *

curSel=ExtAPI.SelectionManager.CurrentSelection

# Need to have at least one node selected, otherwise pass"
if (curSel.Ids.Count==0):
    mess_line1 = 'Please select at least one node'
    MessageBox.Show(mess_line1)

else: # assuming nodes are selected

    meshData=ExtAPI.DataModel.Project.Model.Analyses[0].MeshData

    scoping=dfp.Scoping()
    scoping.Ids = curSel.Ids
    scoping.Location=dfp.enums.location.nodal

    anl=ExtAPI.DataModel.AnalysisByName(ExtAPI.DataModel.AnalysisNames[0])
    rstFile=anl.WorkingDir+'file.rst'

    dataSource = dfp.DataSources(rstFile)

    stressOp1 = dfp.operators.result.stress_principal_1()
    stressOp2 = dfp.operators.result.stress_principal_2()
    stressOp3 = dfp.operators.result.stress_principal_3()

    stressOp1.inputs.data_sources.Connect(dataSource)
    stressOp2.inputs.data_sources.Connect(dataSource)
    stressOp3.inputs.data_sources.Connect(dataSource)

    stressOp1.inputs.mesh_scoping.Connect(scoping)
    stressOp2.inputs.mesh_scoping.Connect(scoping)
    stressOp3.inputs.mesh_scoping.Connect(scoping)

    stressData1=stressOp1.outputs.fields_container.GetData()[0]
    stressData2=stressOp2.outputs.fields_container.GetData()[0]
    stressData3=stressOp3.outputs.fields_container.GetData()[0]

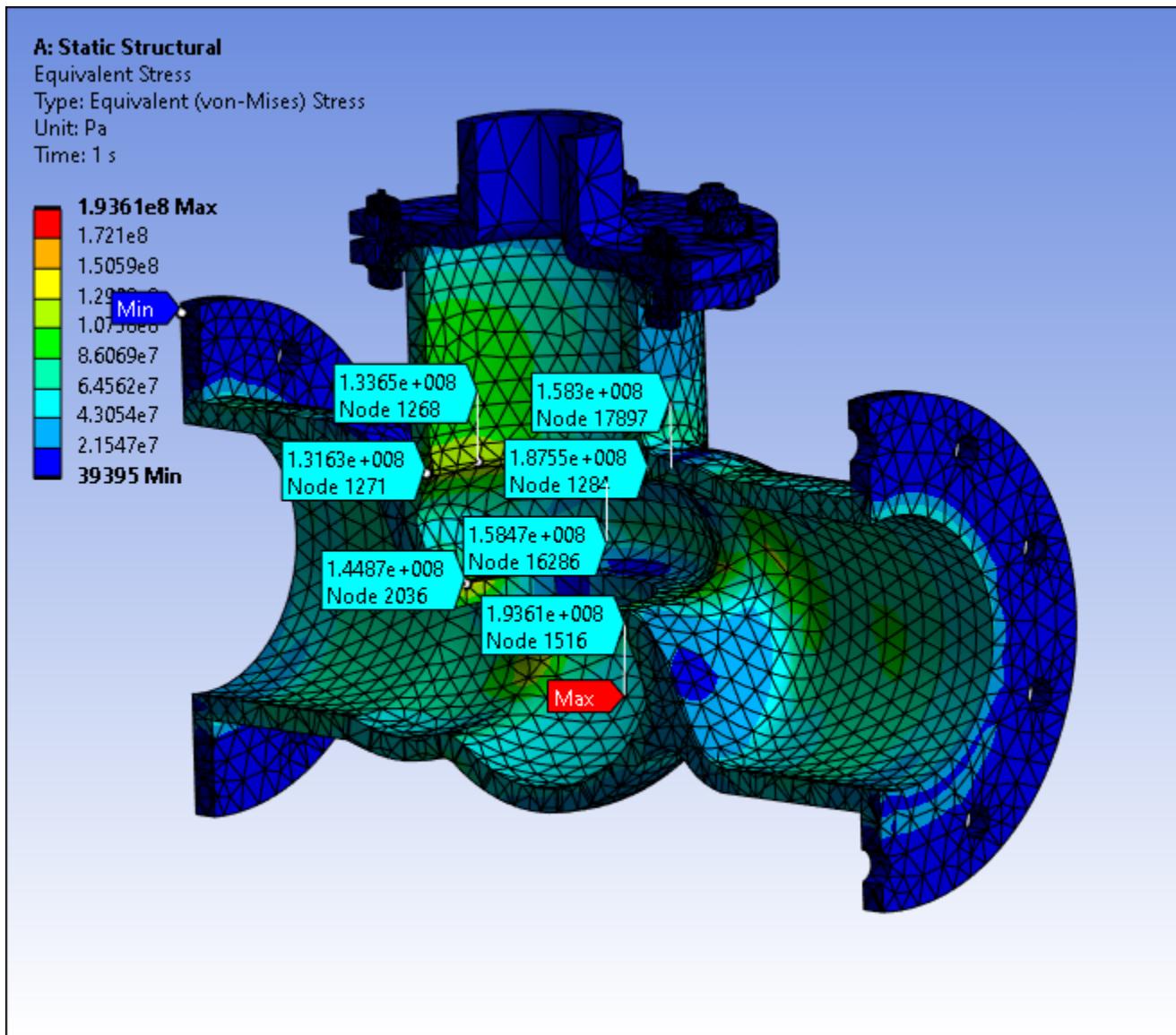
    stressData=[]

    for nid in curSel.Ids:
        stressData.append([nid,stressData1.GetEntityDataById(nid)[0],stressData2.GetEntityDataById(nid)[0],str(stressData3.GetEntityDataById(nid)[0]))]

    # Display result
    mess_line='Node\ts1\ts2\ts3\n'
    for sd in stressData:
        mess_line+=str(sd[0])+'\t'+str(round(sd[1],2))+'\t'+str(round(sd[2],2))+'\t'+str(round(sd[3],2))+'\n'
    MessageBox.Show(mess_line)
```

Finding Hot Spots

For a solved Static Structural that includes an Equivalent Stress result, this script finds the locations where high stresses occur and places a label on them. Here is an illustrated example.



The script specifies the insertion of seven labels. The script places the first label at the result's maximum value, and each subsequent label is placed at a distance of 0.01 (script line 25) in between each node. This distance between labels makes sure that all labels will not be clustered around the maximum value. The unit for this distance is based on what you have selected in your analysis. For the example above, the selected unit is meters. In addition, and again based on the currently selected unit of measure, the result threshold is 20 (script line 27). This means that no result value (or label) below this threshold is included.

```
import mech_dpf
import Ans.DataProcessing as dpf

def nodeDist(cur_node,nodes_dict,test_dist):
    # Compute distance between a reference node (cur_node) and
    # a set of nodes nodes_dict and check vs a test distance (test_dist)
    # if any of the nodes in nodes_dict is close to cur_node within
    # less than test_dist, then return False. Otherwise cur_node is
    # away from each node in nodes_dict by at least test_dist
    test=True
    for nodeId in nodes_dict.keys():
        test_node=nodes_dict[nodeId]
        dist=0.
        for i in range(0,3):
```

```

        dist+=(test_node[i]-cur_node[i])**2
    if (dist<=test_dist**2):
        test=False
        break

    return test

# Number of labels to display
numLabels=7
# Minimal distance between two labels (in current Mechanical unit)
minDistBetweenNodes=0.1
# Minimal stress value for which label should be displayed (in current Mechanical Unit)
thresholdStress=20.

# Dictionaries to store results
maxPairs={}
maxNodes={}

# Retrieve 'Static Structural' analysis - replace analysis name
# based on your current model
analysis=ExtAPI.DataModel.AnalysisByName('Static Structural')
# Retrieve first 'Equivalent Stress' object. replace object name
# based on your current model
resultObject=ExtAPI.DataModel.GetObjectsByName('Equivalent Stress')[0]

# Define datasources (rst from analysis)
dataSource = dpf.DataSources(analysis.ResultFileName)
# Create Mises operator
vmises = dpf.operators.result.stress_von_mises()
vmises.inputs.data_sources.Connect(dataSource)

# retrieve mesh from rst
model=dpf.Model(dataSource)
mesh=model.Mesh

# Min_max operator will be used to identify hotspots
min_max = dpf.operators.min_max.min_max() # operator instantiation

# Temporary operator
currentMises = vmises.outputs.getfields_container()

# Filter used to progressively filter data when maxima are found
tempFilter = dpf.operators.filter.field_low_pass() # operator instantiation

# Filter out stresses below thresholdStress
initFilter = dpf.operators.filter.field_high_pass() # operator instantiation
initFilter.inputs.field.Connect(currentMises)
initFilter.inputs.threshold.Connect(thresholdStress)

# update currentMises with results from filter
currentMises = initFilter.outputs.getfield()

if not(currentMises.Data):
    ExtAPI.Application.LogWarning('Threshold value exceeds maximum stress. Please set a lower value for "thresholds'

# Start looking for hotspots
numFound=0
while numFound<numLabels and currentMises.Data:
    # in current stress field, search for max value
    min_max.inputs.field.Connect(currentMises)
    maxData = min_max.outputs.field_max.GetData()

    # Check whether scoping node is far from other nodes
    if numFound>0:
        # at least one label was found, we need to check for distance
        maxNodeId=maxData.ScopingIds[0]
        maxValue=maxData.Data[0]
        # Retrieve information from node at max value out of result mesh
        node=mesh.NodeById(maxNodeId)
        nodeCoord=[node.X, node.Y,node.Z]
        if nodeDist(nodeCoord,maxNodes,minDistBetweenNodes):

```

```

# node if node is far enough from all other nodes, add to labels
# otherwise, we simply ignore this maximum
maxNodes[maxNodeId]=nodeCoord
maxPairs[maxValue]=maxNodeId
numFound+=1
else:
    # first label at maximum value
    # Retrieve max value and corresponding node from maxData operator
    maxValue=maxData.Data[0]
    maxNodeId=maxData.ScopingIds[0]
    # Start population data in maxPairs (maxvalue:node id)
    # and maxNodes (node id: x,y,z)
    maxPairs[maxValue]=maxNodeId
    node=mesh.NodeById(maxNodeId)
    maxNodes[maxNodeId]=[node.X, node.Y, node.Z]
    numFound+=1

    # Filter out anything above last max value from currentMises
    tempFilter.inputs.field.Connect(currentMises)
    tempFilter.inputs.threshold.Connect(maxValue)
    currentMises = tempFilter.outputs.getfield()

if numFound<numLabels and numFound>0:
    ExtAPI.Application.LogWarning('Could not find more than '+str(numFound)+' hotspots. You should set "thresholdStress" variable or "minDistance" variable')

if numFound==0:
    ExtAPI.Application.LogWarning('Could not find any hotspots. You should set "thresholdStress" variable or "minDistance" variable')
else:
    # Create labels on result object
    with Transaction():
        labelsForRes = Graphics.LabelManager.GetObjectLabels(resultObject)
        Graphics.LabelManager.DeleteLabels(labelsForRes)
        for node in maxPairs.values():
            probeLabel = Graphics.LabelManager.CreateProbeLabel(resultObject)
            probeLabel.Scoping.Node = node

```


Examples Using the Python Code Object

The following script examples are for use with the [Python Code](#) object:

[After Object Changed Filtering](#)

[Send a Warning Message](#)

[Export a Boundary Condition](#)

[Update Following Mesh Generation](#)

[Check Property Values](#)

[Change Material Property](#)

[Print Message to ACT Log](#)

[Property Provider Example](#)

[Parameterize Coordinate System Transformation](#)

[Specify and Parameterize Bolt Pretension Loads](#)

After Object Changed Filtering

Goal

Goal: This Python script checks to see if the object that was changed is for a given category. Specifically, it checks whether a Pressure object was changed and if so, it prints its name to the log file.

The script drops any events that are not associated with the Pressure object. To view the log file, from the Workbench **Project** page, select **Extensions > View Log File**.

Code

```
def after_object_changed(this, object_changed, property_name):# Do not edit this line
    """
    Called after an object is changed.
    Keyword Arguments :
        this -- the datamodel object instance of the python code object you are currently editing in the tree
        object_changed -- The object that was changed
        property_name -- The property that was changed
    """

    # To access properties created using the Property Provider, please use the following command.
    # this.GetCustomPropertyByPath("your_property_group_name/your_property_name")

    # To access scoping properties use the following to access geometry scoping and named selection respective
    # this.GetCustomPropertyByPath("your_property_group_name/your_property_name/Geometry Selection")
    # this.GetCustomPropertyByPath("your_property_group_name/your_property_name/Named Selection")

    # If you are interested in when pressure changes in any way, you can ignore all other object changed even
    # using the line below.
    if object_changed.DataModelObjectCategory != DataModelObjectCategory.Pressure:
        return

    ExtAPI.Log.WriteMessage("Pressure : " + str(property_name) + " changed.")
```

```
pass
```

Send a Warning Message

Goal

The following script sends a warning message if a contact has been changed. It retrieves the contact's name, changed property and new value.

Code

```
def after_object_changed(this, object_changed, property_name):# Do not edit this line
    """
    Called after an object is changed.
    Keyword Arguments :
        this -- this python object
        object_changed -- The object that was changed
        property_name -- The property that was changed
    """

    # This script sends a warning message if a contact has been changed
    # It retrieves the contact name, changed property and new value

    # The script is working with 'after object changed' callback
    # The Python code object may be located under model, analysis or solution

    # The If statement allows to act only when a certain type of object has been changed
    if (object_changed.GetType() == Ansys.ACT.Automation.Mechanical.Connections.ContactRegion):
        msg = Ansys.Mechanical.Application.Message('You changed the following object : '+object_changed.Name+
            '\n Property '+property_name+' has been set to '+str(getattr(object_changed,property_name)), MessageSeverity.Warning)
        ExtAPI.Application.Messages.Add(msg)
```

Export a Boundary Condition

Goal

This script demonstrates how to export displacements to a text file once the simulation is solved. Output of the script is an ASCII file with node number, XYZ location and UX/UY/UZ values over the entire model.

Code

```
def after_post(this, solution):# Do not edit this line
    """
    Called after post processing.
    Keyword Arguments :
        this -- this python object
        solution -- Solution
    """

    # This script shows how to export displacements in a text file once a simulation has been run
    # Output of the script is an ASCII file with node number, XYZ location and UX/UY/UZ values over
    # the entire model

    # The script is working with 'after_post' or 'after_solve' callback
    # You may place the Python code either under the analysis or under solution

    # We are using dpf (Data Processing Framework) for results extraction
    import mech_dpf
    import Ans.DataProcessing as dpf
```

```

import os

solution=ExtAPI.DataModel.GetObjectsByName('Solution')[0]

# Get analysis and retrieve working directory
analysis=solution.Parent
rstFile = os.path.join(analysis.WorkingDir,'file.rst')

# Define rst file as source of data for DPF
dataSource = dpf.DataSources(rstFile)

# Now create displacement operator and connect to datasource
displacements = dpf.operators.result.displacement()
displacements.inputs.data_sources.Connect(dataSource)

# We assume a static analysis with one load step, retrieve actual deformation data from rst
disp=displacements.outputs.fields_container.GetData()[0]

# Get Mesh information to retrieve node coordinates
meshData=ExtAPI.DataModel.Project.Model.Analyses[0].MeshData
nodeIds=disp.ScopingIds # Get node list from results

# Loop over nodes and write to txt file in user_files directory from project
outFileName=os.path.join(analysis.WorkingDir,'..\..\..\user_files\exported_disp.txt')

f_out=open(outFileName,'w')
f_out.write('# Node \t X\tY\tZ\tUY\tUZ\n')
for nid in sorted(nodeIds):
    node=meshData.NodeById(nid) # Node data for nid
    nodeDisp=disp.GetEntityDataById(nid) # displacement from node nid
    f_out.write(str(nid)+'\t'+str(node.X)+'\t'+str(node.Y)+'\t'+str(node.Z)+'
    '\t'+str(nodeDisp[0])+'\t'+str(nodeDisp[1])+'\t'+str(nodeDisp[2])+'\n')
f_out.close()

```

Update Following Mesh Generation

Goal

The following script demonstrates how to automatically update all imported pressures after the mesh has been generated.

Code

```

def after_mesh_generated(this,mesh_object):# Do not edit this line
"""
Called after mesh has been generated.
Keyword Arguments :
    this -- this python object
    mesh_object -- The mesh object that was generated
"""

# This script shows how to automatically update all imported pressures
# after the mesh has been generated

# The script is working with 'after mesh generated' callback
# The Python code object should be located under the model

# Note: the script may not work if you are using shared licenses

# Once the mesh has been generated, get all Imported Pressures to automatically reimport them
# Be careful this could be a time consuming task if many imported loads need to be updated
importedLoads=ExtAPI.DataModel.GetObjectsByType(DataModelObjectCategory.ImportedPressure)

# Loop over all imported pressure and import load
for ild in importedLoads:

```

```
    ild.Activate()
    ild.ImportLoad()
```

Check Property Values

Goal

The following script demonstrates how to check a pressure quantity and the material of a body.
The script outputs a message in the Mechanical **Message** pane.

Code

```
def before_solve(this, analysis):# Do not edit this line
    """
    Called before solving the parent analysis.
    Keyword Arguments :
        this -- this python object
        analysis -- Static Structural
    """

    # This script shows how to check a pressure quantity and body material
    # Output of the script is a messages in Mechanical messages

    # The script is working with 'before_solve' callback
    # The Python code object should be located under the analysis

    # Check if pressure value is 1 MPa
    pres=ExtAPI.DataModel.GetObjectsByName('Pressure')[0] # Replace 'pressure' with the name of an existing lo
    if pres.Magnitude != Quantity(1.0,'MPa'):
        msg = Ansys.Mechanical.Application.Message('Pressure value is not correct - should be 1MPa', MessageSeverityType.W
        ExtAPI.Application.Messages.Add(msg)

    # Check body material
    body=ExtAPI.DataModel.GetObjectsByName(r'Component1\body')[0] # Replace 'Component1\body' with the name of f
    msg = Ansys.Mechanical.Application.Message('Main body material is : '+body.Material, MessageSeverityType.W
    ExtAPI.Application.Messages.Add(msg)
```

Change Material Property

Code

The following script demonstrates how to change a material property for a given body using get body commands.

Code

```
"""
To insert command just after material definitions in /PREP7, use solver_input_file.WriteLine("!Your command")
Global Helpers:
    this -- this python object
    solver_input_file -- file stream that allows you to inject commands into the solver input file
    solver_data -- data structure that allows you to access information from the model such as current step, co
"""

# This script shows how to change a material property for a given body

# The script is working with 'Get Body Commands'
# the Python code is place under the body you want to modify

solver_input_file.WriteLine("! Changing Young modulus for body: "+this.Parent.Name) # Just a comment in the ds
```

```

body_id=this.Parent.GetGeoBody().Id # retrieve the body in the tree with this.Parent, then get corresponding g
matid=solver_data.GetMaterialSolverId(body_id) # This is the material Id for APDL

# Change Young's modulus of this body
solver_input_file.WriteLine("mp,ex,"+str(matid)+",70000")

```

Print Message to ACT Log

Goal

The following script demonstrates how to print debug messages to the ACT log. For more information on how to access the ACT log, see the [Debug Mode](#) section of the *ACT Developer's Guide*.

Code

```

def after_object_changed(this, object_changed, property_name):# Do not edit this line
    """
    Called after an object is changed.
    Keyword Arguments :
        this -- the datamodel object instance of the python code object you are currently editing in the tree
        object_changed -- The object that was changed
        property_name -- The property that was changed
    """

    # To access properties created using the Property Provider, please use the following command.
    # this.GetCustomPropertyByPath("your_property_group_name/your_property_name")

    # To access scoping properties use the following to access geometry scoping and named selection respectively
    # this.GetCustomPropertyByPath("your_property_group_name/your_property_name/Geometry Selection")
    # this.GetCustomPropertyByPath("your_property_group_name/your_property_name/Named Selection")

    # Write debug messages to the extension log using the following command
    ExtAPI.Log.WriteMessage("Inside after_object_changed")

    ExtAPI.Log.WriteMessage("object_changed : " + object_changed.Name)
    ExtAPI.Log.WriteMessage("property_name : " + property_name)

    pass

```

Property Provider Example

Goal

This example has two parts: the Python entry and the Property Provider entry.

- **Python:** This part of the script retrieves all Python Code object properties and writes them to the ds.dat file as comments.
- **Property Provider:** The code contained in the Property Provider tab defines property definitions. The example below creates properties that are similar to what you would see in a Force object.

Python Code

```

"""
To insert commands just prior to the Ansys SOLVE command, use solver_input_file.WriteLine("!Your command")
Global Helpers:
    this -- the datamodel object instance of the python code object you are currently editing in the tree

```

```
solver_input_file -- file stream that allows you to inject commands into the solver input file
solver_data -- data structure that allows you to access information from the model such as current step, co
"""

# To access properties created using the Property Provider, please use the following command.
# this.GetCustomPropertyByPath("your_property_group_name/your_property_name")

# To access scoping properties use the following to access geometry scoping and named selection respectively:
# this.GetCustomPropertyByPath("your_property_group_name/your_property_name/Geometry Selection")
# this.GetCustomPropertyByPath("your_property_group_name/your_property_name/Named Selection")

properties = this.PropertyProvider.GetProperties()
for prop in properties:
    line = "!" + prop.Name + " " + prop.ValueString
    solver_input_file.WriteLine(line)
```

Property Provider Code

```
def reload_props():

    this.PropertyProvider = None
    # comment the return below to allow the rest of the function definition to be executed and add properties
    return

    #Create a new PropertyProvider Instance called Force
    Force = Provider()

    #Add a new group to called Definition2 to Force
    Definition = Force.AddGroup("Definition2")

    #Add an Expression property "Type" to the Definition2 group and set its value = "Force"
    type = Definition.AddProperty("Type", Control.Expression)
    type.Value = "Force"

    #Create an instance of the custom ForceProperty that was created above and add it to the Definition2 group
    force_prop = Definition.AddProperty("Force Property", "Force")

    #Add an Options property "Applied By" Options property
    applied_by = Definition.AddProperty("Applied By", Control.Options)
    applied_by.Options = {0:"Surface Effect", 1:"Direct"}
    applied_by.Value = 0

    #Add Options property "Suppressed" Options property
    suppressed = Definition.AddProperty("Suppressed", Control.Options)
    suppressed.Options = {0:"Yes", 1:"No"}
    suppressed.Value = 1

    #Add a new group called "Scope" to Force
    Scope = Force.AddGroup("Scope")

    #Add a Scoping Property to the Scope group
    scoping_prop = Scope.AddProperty("Scoping Property", "Scoping", "property_templates")

    #Connects the provider instance back to the object by setting the PropertyProvider member on this, 'this' is
    #current instance of the Python Code object.
    this.PropertyProvider = Force

#region Property Provider Template
from Ansys.ACT.Mechanical.AdditionalProperties import PropertyProviderAdapter
from Ansys.ACT.Mechanical.AdditionalProperties import *

"""

The property_templates module is located in {YOUR_INSTALL_DIR}/DesignSpace/DSPages/Python/mech_templates
"""
```

```

from mech_templates import property_templates
property_templates.set_ext_api(ExtAPI)

class Provider(Ansys.ACT.Interfaces.Mechanical.IPropertyProvider):
    """
    Provider template that implements IPropertyProvider to demonstrate the usage of IPropertyProvider.
    It provides helper methods and classes that manage properties that can be dynamically added to an object.
    """
    # region These are callbacks that as a user you may want to modify to get specific behavior
    def IsValid(self, prop):
        """
        Called when checking the validity of a property, with the property instance.
        """

        # for double property use the ValidRange property to check validity
        if(isinstance(prop, DoubleProperty)):
            return prop.ValidRange[0] <= prop.Value and prop.ValidRange[1] >= prop.Value

    return True

    def IsReadOnly(self, prop):
        """
        Called when checking if a property should be readonly, with the property instance.
        """

        return False

    def IsVisible(self, prop):
        """
        Called when checking if a property should be visible, with the property instance.
        """

        return True

    def SetValue(self, prop, val):
        """
        Allows you to override the setter of the Value property on the property instance.
        Keyword Arguments:
            prop -- property of which the value is being set
            val -- the value that was set
        Returns:
            The value that the Value property should be set to
        """
        return val

    def GetValue(self, prop, val):
        """
        Allows you to override the getter of the Value property on the property instance.
        Keyword Arguments:
            prop -- property of which the value is being set
            val -- current value of the Value property
        Returns:
            The value that the getter on the internal value should return
        """
        return val
    # endregion

    #structures that hold property instances
    prop_list = []
    prop_map = {}
    prop_groups = set()

    class AnsGroup():
        """Helper group class to group properties, and provides methods to add properties to groups."""
        provider = None

        def __init__(self, name=None, provider=None):
            self.name = name
            self.provider = provider

```

```

def __AddScopingProperty(self, name):
    """
    Adds a scoping property with a given name to this group.

    Keyword Arguments :
        name -- unique name for the scoping property
    """

    scoping_prop = property_templates.ScopingProperty(name, self.name)

    for prop in scoping_prop.GetGroupedProps():
        self.provider.AddProperty(prop)
    return scoping_prop.GetGroupedProps()

def __AddForceProperty(self, name):
    """
    Adds a Force property with a given name to this group.

    Keyword Arguments :
        name -- unique name for the force property
    """

    force_prop = ForceProperty(name, self.name)

    for prop in force_prop.GetGroupedProps():
        self.provider.AddProperty(prop)
    return force_prop.GetGroupedProps()

def AddProperty(self, name=None, prop_control=None, module_name=None):
    """
    Creates an instance of the property and connects delegates in
    the associated Property Propvider.

    Keyword Arguments :
        name -- unique name for the scoping property
        prop_control -- one of the built in controls, or extended controls
        module_name -- module where the control is defined
    """

    #special case for scoping property
    if(prop_control == "Scoping" and module_name == "property_templates"):
        return self.__AddScopingProperty(name)

    #special case for Force property
    if(prop_control == "Force"):
        return self.__AddForceProperty(name)

    #if no module_name is passed, use the globals in current module
    #that has the built in controls imported

    prop_mod_globals = None

    if(module_name != None):
        if(module_name not in globals()):
            raise Exception("Unknown module : " + module_name)

        prop_mod_globals = globals()[module_name].get_globals()

    else:
        prop_mod_globals = globals()

    #class name is built based on control + "Property"
    #    Double -> DoubleProperty
    prop_class_name = str(prop_control) + "Property"

    if(prop_class_name not in prop_mod_globals):
        raise Exception("Unknown property class : " + prop_class_name)

```

```

#instantiate the property based on module and class name
prop = prop_mod_globals[prop_class_name](self.name + "/" + name, self.name)

if(prop == None):
    raise Exception("Issue while creating the property instance.")

#set the delegates to property provider functions
prop.IsValidCallback = self.provider.IsValid
prop.IsReadOnlyCallback = self.provider.IsReadOnly
prop.IsVisibleCallback = self.provider.IsVisible
prop.GetValueCallback = self.provider.GetValue
prop.SetValueCallback = self.provider.SetValue

#as a default make the property name the property display name
prop.DisplayName = name

#add property to the provider
self.provider.AddProperty(prop)

return prop

def __init__(self):
    pass

def GetProperties(self):
    """
    Returns a list of properties in the order that they were added to the property provider.
    """
    return [self.prop_map[propName] for propName in self.prop_list]

def AddGroup(self, name=None):
    """
    Creates an instance of helper group class and returns it.
    """

    if name in self.prop_groups:
        raise Exception("Group with name " + name + " already exists, please use a unique group name.")

    #keep groups names so we can make sure no duplicate groups are added
    self.prop_groups.add(name)

    return self.AnsGroup(name, self)

def AddProperty(self, prop):
    """
    Method used by the helper group class to add the property to the data-structure holding
    the property instances.
    """

    if(prop.Name in self.prop_map):
        raise Exception("Property name must be unique, property with name '" + prop.Name + "' already exists")

    self.prop_list.append(prop.Name)
    self.prop_map[prop.Name] = prop

#end region

"""
Create a Force like object using PropertyProvider
"""

class ForceProperty:
    """
    Class to create a custom Force property using a combination of different control types
    """
    grouped_props = []
    def __init__(self, name, groupName="Forcing"):

```

```

#Create a DefineBy property using the OptionsProperty
self.def_by_prop = OptionsProperty(name + "/Define By", groupName)
self.def_by_prop.Options = {0: "Vector", 1: "Components"}
self.def_by_prop.Value = 0
self.def_by_prop.DisplayName = ExtAPI.Application.GetLocalString("ID_ScopingDefineBy")

#Create a Magnitude property using DoubleProperty with a valid range.
#Override the default IsVisible and IsValid callbacks with the ones defined int this class.
self.magnitude = DoubleProperty(name + "/Magnitude", groupName)
self.magnitude.Value = 0
self.magnitude.ValidRange = (1.0, 4.0)
self.magnitude.DisplayName = ExtAPI.Application.GetLocalString("ID_MagnitudeProperty")
self.magnitude.IsVisibleCallback = self.Visible
self.magnitude.IsValidCallback = self.IsValid

#Create a Direction property using GeometrySelectionProperty from property_templates.
#Override the default IsVisible callback with the one defined int this class."
self.direction = property_templates.GeometrySelectionProperty(name + "/Direction", groupName)
self.direction.DisplayName = ExtAPI.Application.GetLocalString("ID_Direction")
self.direction.IsVisibleCallback = self.Visible

#Create a Coordinate System property using OptionsProperty.
#Add an Option and set the value to the first option.
#Override the default IsVisible callback with the one defined int this class."
self.coordinate_system = OptionsProperty(name + "/Coordinate System", groupName)
self.coordinate_system.Options = {0: "Global Coordinate System"}
self.coordinate_system.Value = 0
self.coordinate_system.DisplayName = ExtAPI.Application.GetLocalString("ID_Coordinate_System")
self.coordinate_system.IsVisibleCallback = self.Visible

#Create an X Component property using DoubleProperty with a valid range.
#Override the default IsVisible and IsValid callbacks with the ones defined int this class.
#Allow this property to be parameterized by setting CanParameterize to True.
self.x_comp = DoubleProperty(name + "/X Component", groupName)
self.x_comp.Value = 0
self.x_comp.ValidRange = (1.0, 4.0)
self.x_comp.DisplayName = ExtAPI.Application.GetLocalString("ID_DirectionalXComponent")
self.x_comp.IsVisibleCallback = self.Visible
self.x_comp.CanParameterize = True
self.x_comp.IsValidCallback = self.IsValid

#Create a Y Component property using DoubleProperty with a valid range.
#Override the default IsVisible and IsValid callbacks with the ones defined int this class.
#Allow this property to be parameterized by setting CanParameterize to True.
self.y_comp = DoubleProperty(name + "/Y Component", groupName)
self.y_comp.Value = 0
self.y_comp.ValidRange = (1.0, 4.0)
self.y_comp.DisplayName = ExtAPI.Application.GetLocalString("ID_DirectionalYComponent")
self.y_comp.IsVisibleCallback = self.Visible
self.y_comp.CanParameterize = True
self.y_comp.IsValidCallback = self.IsValid

#Create a Z Component property using DoubleProperty with a valid range.
#Override the default IsVisible and IsValid callbacks with the ones defined int this class.
#Allow this property to be parameterized by setting CanParameterize to True.
self.z_comp = DoubleProperty(name + "/Z Component", groupName)
self.z_comp.ValidRange = (1.0, 4.0)
self.z_comp.Value = 0
self.z_comp.DisplayName = ExtAPI.Application.GetLocalString("ID_DirectionalZComponent")
self.z_comp.IsVisibleCallback = self.Visible
self.z_comp.CanParameterize = True
self.z_comp.IsValidCallback = self.IsValid

#Add all properties to the grouped_props list
self.grouped_props = [self.def_by_prop, self.magnitude, self.direction, self.coordinate_system, self.x_]

def GetGroupedProps(self):
    """
    Returns the properties list

```

```

"""
    return self.grouped_props

def IsVisible(self, prop):
    """
    Overrides the IsVisible callback for properties.
    Decides which properties will be visible in the details pane based on the option selected in the Definition tab.
    """
    if(prop.DisplayName == "Magnitude"):
        return int(self.def_by_prop.Value) == 0
    if(prop.DisplayName == "Direction"):
        return int(self.def_by_prop.Value) == 0
    if(prop.DisplayName == "Coordinate System"):
        return int(self.def_by_prop.Value) == 1
    if(prop.DisplayName == "X Component"):
        return int(self.def_by_prop.Value) == 1
    if(prop.DisplayName == "Y Component"):
        return int(self.def_by_prop.Value) == 1
    if(prop.DisplayName == "Z Component"):
        return int(self.def_by_prop.Value) == 1

def IsValid(self, prop):
    """
    Overrides the isValid callback for properties.
    """
    #for double property use the ValidRange property to check validity
    if(isinstance(prop, DoubleProperty)):
        return prop.ValidRange[0] <= prop.Value and prop.ValidRange[1] >= prop.Value

    return True

"""

Reload the properties at the end to make sure the class definition is executed before instantiation
"""
reload_props()

```

Parameterize Coordinate System Transformation

Goal

The following script demonstrates how to parameterize a coordinate system transformation.

Code PyCode_OnAfterObjectChanged

```

def after_object_changed(this, object_changed, property_name):# Do not edit this line
    """
    Called after an object is changed.
    Keyword Arguments :
        this -- the datamodel object instance of the python code object you are currently editing in the tree
        object_changed -- The object that was changed
        property_name -- The property that was changed
    """

    # We ignore any changes other than changes made to the this python code object itself
    # We listen for any changes made to this object so we can sync up parameterized properties
    # with the coordinate system, everytime the design point updates happen
    if this.DataModelObjectCategory != object_changed.DataModelObjectCategory:
        return

    my_coordinate_system = ExtAPI.DataModel.GetObjectsByName("my CS")[0]
    coordinate_system_props=my_coordinate_system.Properties

    # Sync up the parameterized properties on this object with the coordinate system
    for prop in coordinate_system_props:
        if prop.Name.find("Transform") != -1:

```

```
newVal=this.GetCustomPropertyByPath("CS Transformations/" + prop.Caption)
prop.InternalValue=newVal
```

PyCode_OnBeforeSolve

```
def before_solve(this, analysis):# Do not edit this line
    """
    Called before solving the parent analysis.
    Keyword Arguments :
        this -- the datamodel object instance of the python code object you are currently editing in the tree
        analysis -- Static Structural
    """

    # Log out the values of properties with the name that contain "Transform"
    # to demonstrate the values for the coordinate system are parameterized

    import os
    import os.path
    sys.path.append(os.getenv("TB_WORKING_DIRECTORY"))
    WDIR = os.getenv("TB_WORKING_DIRECTORY")
    fo2 = open(WDIR + "/DS PYTHON_SNIPPET_019_Generated.log", "a")

    fo2.writelines(" === Scenario Starts here === " + "\n" )
    fo2.writelines("Property Values for Coordinate System set through Parameter Set:" + "\n" )
    my_coordinate_system = ExtAPI.DataModel.GetObjectByName("my CS")[0]
    coordinate_system_props=my_coordinate_system.Properties
    # Log out the values of the properties
    for prop in coordinate_system_props:
        if prop.Name.find("Transform") != -1:
            fo2.writelines(prop.Caption + " = " + str(prop.InternalValue) + "\n")
    fo2.writelines("Property Values for Coordinate System in UI:" + "\n" )
    fo2.writelines(my_coordinate_system.VisibleProperties[17].Caption + " = " + str(my_coordinate_system.VisibleProperties[17].InternalValue) + "\n")
    fo2.writelines(my_coordinate_system.VisibleProperties[18].Caption + " = " + str(my_coordinate_system.VisibleProperties[18].InternalValue) + "\n")
    fo2.writelines(my_coordinate_system.VisibleProperties[19].Caption + " = " + str(my_coordinate_system.VisibleProperties[19].InternalValue) + "\n")
    fo2.writelines(my_coordinate_system.VisibleProperties[20].Caption + " = " + str(my_coordinate_system.VisibleProperties[20].InternalValue) + "\n")
    fo2.writelines(my_coordinate_system.VisibleProperties[21].Caption + " = " + str(my_coordinate_system.VisibleProperties[21].InternalValue) + "\n")
    fo2.writelines(my_coordinate_system.VisibleProperties[22].Caption + " = " + str(my_coordinate_system.VisibleProperties[22].InternalValue) + "\n")
    fo2.writelines(my_coordinate_system.VisibleProperties[23].Caption + " = " + str(my_coordinate_system.VisibleProperties[23].InternalValue) + "\n")
    fo2.writelines(my_coordinate_system.VisibleProperties[24].Caption + " = " + str(my_coordinate_system.VisibleProperties[24].InternalValue) + "\n")
    fo2.writelines(my_coordinate_system.VisibleProperties[25].Caption + " = " + str(my_coordinate_system.VisibleProperties[25].InternalValue) + "\n")
    fo2.writelines(" === Scenario Ends here === " + "\n" )
    fo2.close()
```

Specify and Parameterize Bolt Pretension Loads

Goal

The following script demonstrates how to automatically set and parametrize the Preload value of all the Bolt Pretension loads that are present in the analysis. Note that these loads need to have the same name to be updated by the script. See an [example \(p. 223\)](#) of the executed script below.

Code

```
def after_object_changed(this, object_changed, property_name):# Do not edit this line
    """
    Called after an object is changed.
    Keyword Arguments :
        this -- the datamodel object instance of the python code object you are currently editing in the tree
        object_changed -- The object that was changed
        property_name -- The property that was changed
    """

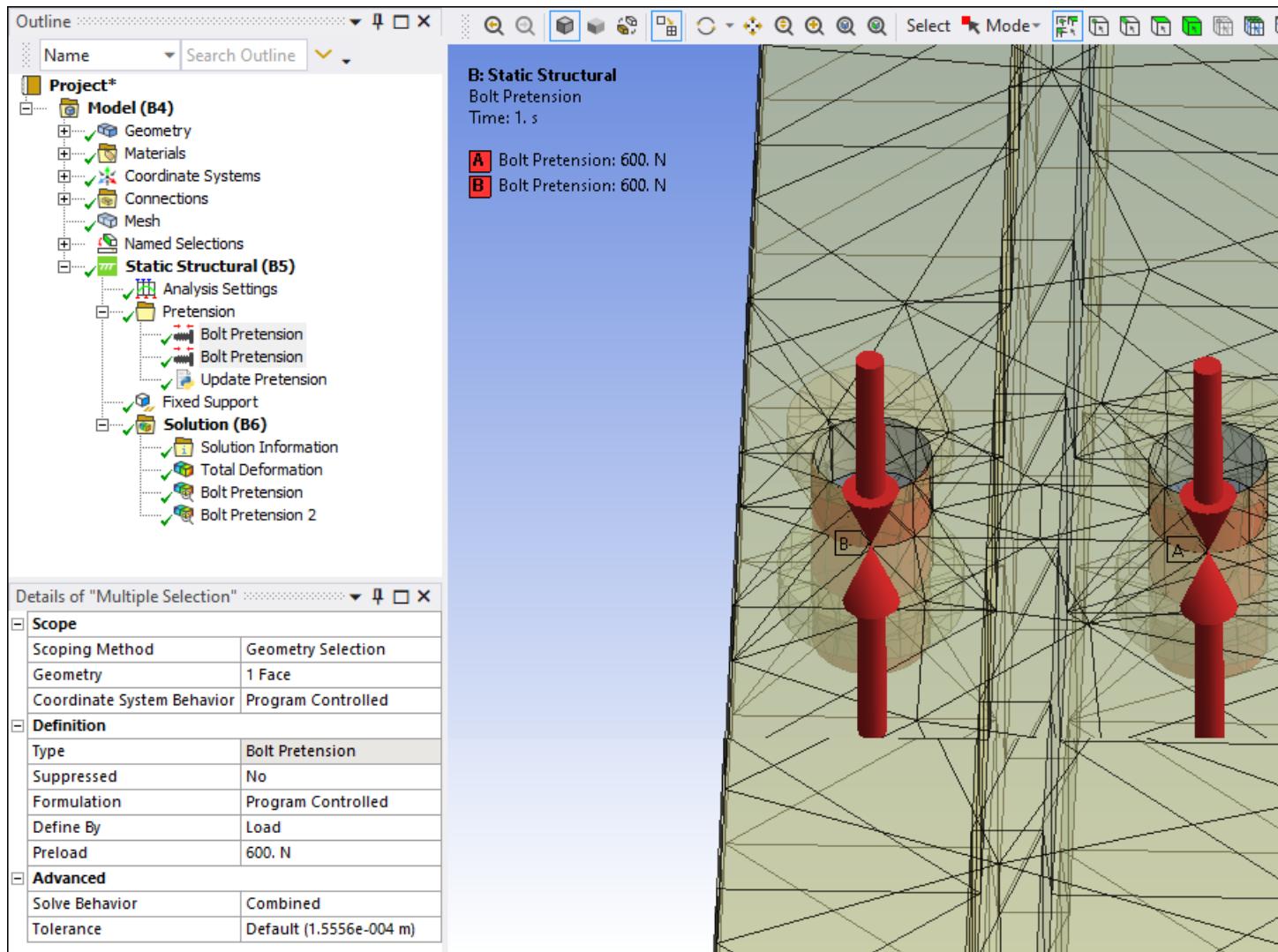
    # To access properties created using the Property Provider, please use the following command.
```

```
# this.GetCustomPropertyByPath("your_property_group_name/your_property_name")  
  
# To access scoping properties use the following to access geometry scoping and named selection respective  
# this.GetCustomPropertyByPath("your_property_group_name/your_property_name/Geometry Selection")  
# this.GetCustomPropertyByPath("your_property_group_name/your_property_name/Named Selection")  
  
if (object_changed.Name != this.Name):  
    # The change is not on the Python code object, simply ignore  
    return  
  
preload = this.GetCustomPropertyByPath("Parameters/Preload (N)").Value  
  
all_prets = DataModel.GetObjectsByName('Bolt Pretension')  
  
for prets in all_prets:  
    times = range(1,prets.Parent.AnalysisSettings.NumberOfSteps+1)  
    preload_vals=[]  
    time_vals=[]  
    for time in times:  
        time_vals.append(Quantity(float(time), "sec"))  
        preload_vals.append(Quantity(float(preload), "N"))  
  
    prets.Preload.Inputs[0].DiscreteValues = time_vals  
    prets.Preload.Output.DiscreteValues=preload_vals
```

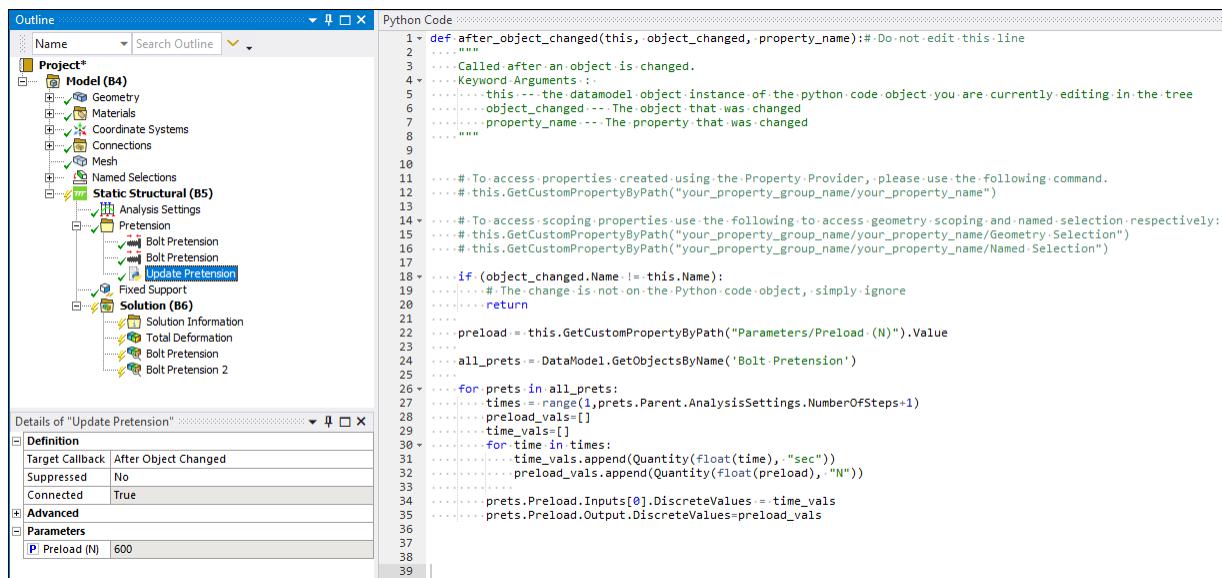
Example

Here you can see the specified loading conditions.

Examples Using the Python Code Object



Here is an example of the script with the corresponding Details of the Python object.



And here is a portion of the code that is included in the Property Provider pane for the example.

```
def reload_props():
    this.PropertyProvider = None
    """
    Sample code is provided below that shows how to:

    1. Create an instance of the Provider. The Provider class is used to add custom properties to the data
    2. Use the Provider instance to add custom properties.
    3. Configure those custom properties.

    """

    # Create the property instance
    provider = Provider()

    # Create a group named Group 1.
    group = provider.AddGroup("Parameters")

    # Create a property with control type Expression and a property with control type Double, and add it to the group
    double_prop = group.AddProperty("Preload (N)", Control.Double)

    # Configure the double property to be parameterizable. As a default the property will be an input parameter
    # However, by updating the ParameterType property, it can be configured to be a output parameter as well.
    double_prop.CanParameterize = True
    double_prop.ParameterType = ParameterType.Input
    # Connects the provider instance back to the object by setting the PropertyProvider member on this, 'this'
    # current instance of the Python Code object.
    this.PropertyProvider = provider
```


End-to-End Analysis Examples

The following examples describe a sequence of APIs that perform a complete analysis for the given analysis scenario.

Note:

These examples assume that you have the automatic contact detection option turned on.

Structural Analyses	Fracture Analyses	Rigid Dynamic Analyses
Static Structural (p. 228)	Semi-Elliptical Crack (p. 267)	Contact Specification (p. 300)
Static Structural Spatially Varying Load (p. 230)	SMART Crack Growth (p. 270)	Flexible Part Specification (p. 302)
Transient Structural (p. 232)	Contact Debonding (p. 276)	General Analysis (p. 305)
Static Structural General Joint (p. 236)	Interface Delamination (p. 279)	Joint Specification (p. 307)
Static Structural Universal Joint (p. 238)		Variable Load Specification (p. 309)
Static Structural Cylindrical Joint (p. 240)		
Symmetry Analyses	Acoustics Analyses	Electric and Magnetic Analyses
Static Structural Symmetric Symmetry (p. 243)	Harmonic Acoustics (p. 282)	Steady State Electric Conduction (p. 312)
Static Structural Linear Periodic Symmetry (p. 246)	Modal Acoustics (p. 285)	Steady-State Thermal-Electric Conduction (p. 315)
Static Structural Cyclic Symmetry (p. 249)		Magnetostatic (p. 317)
Coupled Field Analysis	Structural Optimization Analyses	Thermal Analyses
Coupled Field Static (p. 258)	Density Based Method (p. 287)	Steady-State Thermal Analysis (p. 254)
Coupled Field Harmonic (p. 260)	Level Set Based Method (p. 291)	Transient Thermal Analysis (p. 254)
Coupled Field Transient (p. 263)	Lattice Optimization (p. 294)	
Coupled Field Modal (p. 265)		

Miscellaneous Analyses/Features	External Model
Post Processing Options (p. 320)	Shape Optimization (p. 297)
Post Processing User Defined Results (p. 322)	Static Structural Analysis using External Model (p. 330)
Random Vibration Analysis (p. 324)	

Static Structural Analysis

In this example, using the support files, you will insert a Static Structural analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will define and solve the analysis.

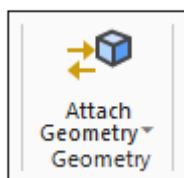
This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Mechanical_Static_Structural_Example_001_Geometry.agdb
- Mechanical_Static_Structural_Example_001_Script.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Static Structural** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the **Geometry Context tab**. Navigate to the proper folder location and select `Mechanical_Static_Structural_Example_001_Geometry.agdb`.



4. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.
5. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select `Mechanical_Static_Structural_Example_001_Script.py`.

6. Select the **Run Script** option ( from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Section 1 - Set up the Tree Object Items.

CS_GRP = Model.CoordinateSystems
STAT_STRUC = Model.Analyses[0]
ANALYSIS_SETTINGS = STAT_STRUC.Children[0]
SOLN= STAT_STRUC.Solution

# Section 2 - Set up the Unit System.

ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS
ExtAPI.Application.ActiveAngleUnit = AngleUnitType.Radian

# Section 3 Named Selection and Coordinate System.

NS1 = Model.NamedSelections.Children[0]
NS2 = Model.NamedSelections.Children[1]
NS3 = Model.NamedSelections.Children[2]
NS4 = Model.NamedSelections.Children[3]
GCS = CS_GRP.Children[0]
LCS1 = CS_GRP.Children[1]

# Section 4 Define remote point.

RMPT_GRP = Model.RemotePoints
RMPT_1 = RMPT_GRP.AddRemotePoint()
RMPT_1.Location = NS1
RMPT_1.XCoordinate=Quantity("7 [m]")
RMPT_1.YCoordinate=Quantity("0 [m]")
RMPT_1.ZCoordinate=Quantity("0 [m]")

# Section 5 - Define Mesh Settings.

MSH = Model.Mesh
MSH.ElementSize =Quantity("0.5 [m]")
MSH.GenerateMesh()

# Section 6 Define boundary conditions.

# Insert FIXED Support
FIX_SUP = STAT_STRUC.AddFixedSupport()
FIX_SUP.Location = NS2

# Insert Frictionless Support
FRIC_SUP = STAT_STRUC.AddFrictionlessSupport()
FRIC_SUP.Location = NS3

# Section 7 - Define remote force.

REM_FRC1 = STAT_STRUC.AddRemoteForce()
REM_FRC1.Location = RMPT_1
REM_FRC1.DefineBy =LoadDefineBy.Components
REM_FRC1.XComponent.Output.DiscreteValues = [Quantity("1e10 [N]")]
REM_FRC1.YComponent.Output.DiscreteValues = [Quantity("1e10 [N]")]
REM_FRC1.ZComponent.Output.DiscreteValues = [Quantity("1e10 [N]")]

# Section 8 - Define thermal condition.

THERM_COND = STAT_STRUC.AddThermalCondition()
THERM_COND.Location = NS4
THERM_COND.Magnitude.Output.DefinitionType=VariableDefinitionType.Formula
THERM_COND.Magnitude.Output.Formula="50*(20+z)"
THERM_COND.XYZFunctionCoordinateSystem=LCS1
THERM_COND.RangeMinimum=Quantity("-20 [m]")
THERM_COND.RangeMaximum=Quantity("20 [m]")
THERM_COND.ThermalConductionType=ConductionType.Conduction
```

```
THERM_COND.RangeMaximum=Quantity("1 [m]")

# Section 9 - Insert directional deformation.

DIR_DEF = STAT_STRUC.Solution.AddDirectionalDeformation()
DIR_DEF.Location = NS1
DIR_DEF.NormalOrientation =NormalOrientationType.XAxis

# Section 10 - Add Total Deformation and force reaction probe

TOT_DEF = STAT_STRUC.Solution.AddTotalDeformation()

# Add Force Reaction
FRC_REAC_PROBE = STAT_STRUC.Solution.AddForceReaction()
FRC_REAC_PROBE.BoundaryConditionSelection = FIX_SUP
FRC_REAC_PROBE.ResultSelection =ProbeDisplayFilter.XAxis

# Section 11 - Solve and get the results.

# Solve Static Analysis
STAT_STRUC.Solution.Solve(True)

DIR_DEF_MAX = DIR_DEF.Maximum.Value
DIR_DEF_MIN = DIR_DEF.Minimum.Value
TOT_DEF_MAX = TOT_DEF.Maximum.Value
TOT_DEF_MIN = TOT_DEF.Minimum.Value
FRC_REAC_PROBE_VAL = FRC_REAC_PROBE.XAxis.Value
```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Static Structural Spatially Varying Load Analysis

In this example, using the support file, you will insert a Static Structural analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will define a spatially varying pressure load and solve the analysis.

This example begins in the Mechanical application. It requires you to download the following files:

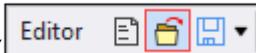
- Static_Structural_Spatially_Varying.agdb
- Static_Structural_Spatially_Varying.py

These files are available [here](#).

Procedure

1. Open Workbench and insert a **Static Structural** system into the **Project Schematic**.
2. Right-click the **Geometry** cell and select **Properties**.
3. Enable the **Named Selections** check box under the **Basic Geometry Options** category.
4. Right-click the **Geometry** cell and select **Import Geometry > Browse** and then navigate to the proper folder location and select *Static_Structural_Spatially_Varying.agdb*.
5. Open Mechanical: right-click the **Model** cell and select **Edit**.

6. Select the Automation tab and select the **Scripting** option to open the **Mechanical Scripting** pane.

7. Select the **Open Script** option ( from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select `Static_Structural_Spatially_Varying.py`.

8. Select the **Run Script** option () from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
#Scenario 1: Store main Tree Object items
MODEL = Model
GEOM = MODEL.Geometry
COORDINATE_SYSTEMS = Model.CoordinateSystems
MESH = Model.Mesh
NAMED_SELECTIONS = Model.NamedSelections

PARTS = GEOM.GetChildren(DataModelObjectCategory.Part, False)
for part in PARTS:
    bodies = part.GetChildren(DataModelObjectCategory.Body, False)
    for body in bodies:
        if body.Name == "Part 1":
            PART1 = body

STAT_STRUC = DataModel.Project.Model.Analyses[0]
ANALYSIS_SETTINGS = STAT_STRUC.AnalysisSettings
STAT_STRUC SOLUTION = STAT_STRUC.Solution

#Scenario 2: Set Display Unit System
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

#Scenario 3: Set isometric view and zoom to fit
cam = Graphics.Camera
cam.SetSpecificViewOrientation(ViewOrientationType.Iso)
cam.SetFit()

#Scenario 4: Store Named Selections for applying load and boundary condition in structural analysis
tot_child_NS_GRP = NAMED_SELECTIONS.Children.Count
for i in range(0, tot_child_NS_GRP, 1):
    ns = NAMED_SELECTIONS.Children[i].Name
    if(ns=='NS_FACE1'):
        NS_FACE1 = NAMED_SELECTIONS.Children[i]
    if(ns=='NS_FACE2'):
        NS_FACE2 = NAMED_SELECTIONS.Children[i]
    if(ns=='NS_FACE3'):
        NS_FACE3 = NAMED_SELECTIONS.Children[i]

#Scenario 5: Store Coordinate System for applying spatially varying load
COORDINATE_SYSTEM = COORDINATE_SYSTEMS.Children[1]

#Scenario 6: Define global mesh size and generate mesh
MESH.Activate()
MESH.ElementSize = Quantity('0.001 [m]')
MESH.GenerateMesh()

#Scenario 7: Define load and boundary conditions in Static Structural analysis
# Add Fixed Support
FIXED_SUPPORT = STAT_STRUC.AddFixedSupport()
FIXED_SUPPORT.Location = NS_FACE1

# Add spatially varying Pressure based on z coordinate of local coordinate system
PRESSURE = STAT_STRUC.AddPressure()
PRESSURE.Location = NS_FACE3
```

```
PRESSURE.Magnitude.Output.Formula = "1e9*z"
PRESSURE.XYZFunctionCoordinateSystem = COORDINATE_SYSTEM

#Scenario 8: Add results in Static Structural analysis
STAT_STRUC SOLUTION.Activate()
TOTAL_DEFORMATION = STAT_STRUC SOLUTION.AddTotalDeformation()
DIRECTIONAL_DEFORMATION = STAT_STRUC SOLUTION.AddDirectionalDeformation()
DIRECTIONAL_DEFORMATION.NormalOrientation = NormalOrientationType.YAxis
DIRECTIONAL_DEFORMATION.CoordinateSystem = COORDINATE_SYSTEM

FORCEREACTION_PROBE = STAT_STRUC SOLUTION.AddForceReaction()
FORCEREACTION_PROBE.BoundaryConditionSelection = FIXED_SUPPORT
FORCEREACTION_PROBE.Orientation = COORDINATE_SYSTEM

MOMENTREACTION_PROBE = STAT_STRUC SOLUTION.AddMomentReaction()
MOMENTREACTION_PROBE.BoundaryConditionSelection = FIXED_SUPPORT
MOMENTREACTION_PROBE.Orientation = COORDINATE_SYSTEM

#Scenario 9: Solve and review Results
STAT_STRUC SOLUTION.Activate()
STAT_STRUC.Solve(True)

#Total Deformation Result
TOTAL_DEF_MIN = TOTAL_DEFORMATION.Minimum.Value
TOTAL_DEF_MAX = TOTAL_DEFORMATION.Maximum.Value

#Directional Deformation Result
DIRECTIONAL_DEF_MIN = DIRECTIONAL_DEFORMATION.Minimum.Value
DIRECTIONAL_DEF_MAX = DIRECTIONAL_DEFORMATION.Maximum.Value

FORCEREACTION_PROBE_Y=FORCEREACTION_PROBE.YAxis.Value
MOMENTREACTION_PROBE_X=MOMENTREACTION_PROBE.XAxis.Value
MOMENTREACTION_PROBE_TOTAL=MOMENTREACTION_PROBE.Total.Value
```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Transient Structural Analysis

In this example, using the support files, you will insert a Transient Structural analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will define and solve the analysis.

This example begins in the Mechanical application. It requires you to download the following CAD model and python files.

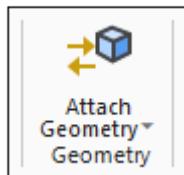
- Transient_Structural_001.x_t
- Transient_Structural_001.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).

2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Transient Structural** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the **Geometry Context tab**. Navigate to the proper folder location and select Transient_Structural_001.x_t.



4. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.



5. Select the **Open Script** option (from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select Transient_Structural_001.py.



6. Select the **Run Script** option (from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Section 1 - Set up the Tree Object Items.
#Section 2 - Set Units.
#Section 3 - Select Frictionless Contact
#Section 4 - Create Named Selection
#Section 5 - Create Mesh Refinement
#Section 6 - Analysis Settings
#Section 7 - Define Boundary Conditions and Loads
#Section 8 - Insert Results
#Section 9 - Solve and Evaluate Results

# Section 1 - Set up the Tree Object Items.

CONT_REG = DataModel.Project.Model.Connections.Children[0]
MSH = DataModel.Project.Model.Mesh
TRANS_STRUC1 = DataModel.AnalysisByName("Transient")
ANA_SETTING1 = TRANS_STRUC1.AnalysisSettings
SOLN= TRANS_STRUC1.Solution

#Section 2 - Set Units

ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS
ExtAPI.Application.ActiveAngleUnit = AngleUnitType.Radian

#Section 3 - Select Frictionless Contact

CONT1 = CONT_REG.Children[0]
CONT1.ContactType = ContactType.Frictionless
CONT1.InterfaceTreatment =ContactInitialEffect.AdjustToTouch

#Section 4 - Create Named Selection

NS1 = DataModel.Project.Model.AddNamedSelection()
NS1.ScopingMethod=GeometryDefineByType.Worksheet
NS1.Name = "MSH_ref_1"
GEN_CRT1 = NS1.GenerationCriteria
```

```

CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
CRT1.Action=SelectionActionType.Add
CRT1.EntityType=SelectionType.GeoFace
CRT1.Criterion=SelectionCriterionType.Size
CRT1.Operator=SelectionOperatorType.Equal
CRT1.Value=Quantity('9397.9 [mm mm]')
GEN_CRT1.Add(CRT1)
NS1.Activate()
NS1.Generate()

NS1.Duplicate()
NS2= DataModel.Project.Model.NamedSelections.Children[1]
NS2.Name = "MSH_ref_2"
GEN_CRT2 = NS2.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
GEN_CRT2[0].Value=Quantity('62653 [mm^2]')
NS2.Activate()
NS2.Generate()

NS1.Duplicate()
NS3= DataModel.Project.Model.NamedSelections.Children[2]
NS3.Name = "cmpsnonlysupp"
GEN_CRT3 = NS3.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
GEN_CRT3[0].Value=Quantity('22555 [mm^2]')
NS3.Activate()
NS3.Generate()

NS1.Duplicate()
NS4= DataModel.Project.Model.NamedSelections.Children[3]
NS4.Name = "frctlsupp"
GEN_CRT4 = NS4.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
GEN_CRT4[0].Value=Quantity('32558 [mm^2]')
CRT2 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT2.Active=True
CRT2.Action=SelectionActionType.Add
CRT2.EntityType=SelectionType.GeoFace
CRT2.Criterion=SelectionCriterionType.Size
CRT2.Operator=SelectionOperatorType.Equal
CRT2.Value=Quantity('60155 [mm mm]')
GEN_CRT4.Add(CRT2)
CRT3 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT3.Active=True
CRT3.Action=SelectionActionType.Add
CRT3.EntityType=SelectionType.GeoFace
CRT3.Criterion=SelectionCriterionType.Size
CRT3.Operator=SelectionOperatorType.Equal
CRT3.Value=Quantity('7764.6 [mm mm]')
GEN_CRT4.Add(CRT3)
NS4.Activate()
NS4.Generate()

NS1.Duplicate()
NS5= DataModel.Project.Model.NamedSelections.Children[4]
NS5.Name = "DISP1"
GEN_CRT5 = NS5.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
GEN_CRT5[0].Value=Quantity('7764.6 [mm^2]')
NS5.Activate()
NS5.Generate()

# Section 5 - Create Mesh Refinement

MSH.ElementSize =Quantity("10 [mm]")
REFINEMENT1 = MSH.AddRefinement()

```

```

SEL=ExtAPI.SelectionManager.AddSelection(NS1)
SEL1=ExtAPI.SelectionManager.CurrentSelection
CLRSEL = ExtAPI.SelectionManager.ClearSelection()
REFINEMENT1.Location = SEL1
REFINEMENT1.NumberOfRefinements = 1

#Section 6 - Analysis Settings

ANA_SETTING1.Activate()
ANA_SETTING1.NumberOfSteps=2
ANA_SETTING1.CurrentStepNumber=1
ANA_SETTING1.StepEndTime=Quantity("1 [sec]")
ANA_SETTING1.InitialTimeStep=Quantity("0.5 [sec]")
ANA_SETTING1.MinimumTimeStep=Quantity("0.5 [sec]")
ANA_SETTING1.MaximumTimeStep=Quantity("0.5 [sec]")
ANA_SETTING1.CurrentStepNumber=2
ANA_SETTING1.StepEndTime=Quantity("2 [sec]")
ANA_SETTING1.InitialTimeStep=Quantity("0.5 [sec]")
ANA_SETTING1.MinimumTimeStep=Quantity("0.5 [sec]")
ANA_SETTING1.MaximumTimeStep=Quantity("0.5 [sec]")

ANA_SETTING1.NewtonRaphsonOption=NewtonRaphsonType.ProgramControlled
ANA_SETTING1.ForceConvergence=ConvergenceToleranceType.On

# Section 7 - Define Boundary Conditions and Loads

COMP_SUP = TRANS_STRUC1.AddCompressionOnlySupport()
COMP_SUP.Location = NS3

BEARING_LD = TRANS_STRUC1.AddBearingLoad()
BEARING_LD.Location = NS2
BEARING_LD.DefineBy = LoadDefineBy.Components
BEARING_LD.XComponent.Inputs[0].DiscreteValues = [Quantity("0 [sec]"), Quantity("1 [sec]"), Quantity("2 [sec]")]
BEARING_LD.XComponent.Output.DiscreteValues = [Quantity("1e5 [N]"), Quantity("2e5 [N]"), Quantity("3e5 [N]")]

FRIC_SUP = TRANS_STRUC1.AddFrictionlessSupport()
FRIC_SUP.Location = NS4

DISP1 = TRANS_STRUC1.AddDisplacement()
DISP1.Location = NS5
DISP1.DefineBy = LoadDefineBy.Components
DISP1.YComponent.Inputs[0].DiscreteValues = [Quantity("0 [sec]"), Quantity("1 [sec]"), Quantity("2 [sec]")]
DISP1.YComponent.Output.DiscreteValues = [Quantity("0.0 [m]"), Quantity("0.0 [m]"), Quantity("0.0 [m]")]

#Section 8 - Insert Results

FRC_REAC_PROBE = TRANS_STRUC1.Solution.AddForceReaction()
FRC_REAC_PROBE.BoundaryConditionSelection = COMP_SUP

EQV_STRS_1 = TRANS_STRUC1.Solution.AddEquivalentStress()

STRS_TOOL_1 = TRANS_STRUC1.Solution.AddStressTool()
STRS_SAF_FCTR_1 = STRS_TOOL_1.AddSafetyFactor()
STRS_TOOL_1.StressLimitType = SafetyLimitType.UltimatePerMaterial

#Section 9 - Solve and Evaluate Results

TRANS_STRUC1.Solution.Solve(True)

EQV_STRS_MAX_1 = EQV_STRS_1.MaximumOfMaximumOverTime.Value
EQV_STRS_MIN_1 = EQV_STRS_1.MinimumOfMinimumOverTime.Value
FRC_REAC_PROBE_VAL = FRC_REAC_PROBE.XAxis.Value
STRS_SAF_FCTR_1_VAL = STRS_SAF_FCTR_1.Minimum.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Static Structural General Joint Analysis

In this example, using the support files, you will insert a Static Structural analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will specify a General Joint as Translational, define mesh and boundary and result objects and solve the analysis.

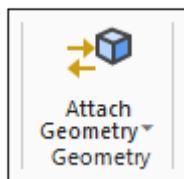
This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Mechanical_Static_General_Joint_Example_Script.x_t
- Mechanical_Static_General_Joint_Example_Script.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Static Structural** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the [Geometry Context tab](#). Navigate to the proper folder location and select `Mechanical_Static_General_Joint_Example_Script.x_t`.



4. Select the [Automation tab](#) and select the **Scripting** option to open the **Mechanical Scripting** pane.

5. Select the **Open Script** option (from the [Editor \(p. 4\)](#) toolbar. Navigate to the proper folder location and select `Mechanical_Static_Structural_Example_001_Script.py`.

6. Select the **Run Script** option (from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
#Scenario 1 - Set up the Units System
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

#Scenario 2 - Set up the Tree Object Items
```

```

CONNECTION_GROUP = Model.Connections
MESH = Model.Mesh
STATIC_STRUCTURAL = Model.Analyses[0]
ANALYSIS_SETTINGS = STATIC_STRUCTURAL.Children[0]
STATIC_STRUCTURAL_SOLUTION = STATIC_STRUCTURAL.Solution

#Scenario 3 - Define Named Selections

NS_FIXED_FACE = DataModel.Project.Model.AddNamedSelection()
NS_FIXED_FACE.ScopingMethod=GeometryDefineByType.Worksheet
NS_FIXED_FACE.Name = "NS_FIXED_FACE"
GEN_CRT1 = NS_FIXED_FACE.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
CRT1.Action=SelectionActionType.Add
CRT1.EntityType=SelectionType.GeoFace
CRT1.Criterion=SelectionCriterionType.LocationZ
CRT1.Operator=SelectionOperatorType.Equal
CRT1.Value=Quantity('0.00 [m]')
GEN_CRT1.Add(CRT1)
NS_FIXED_FACE.Activate()
NS_FIXED_FACE.Generate()

NS_BC_FACE = DataModel.Project.Model.AddNamedSelection()
NS_BC_FACE.ScopingMethod=GeometryDefineByType.Worksheet
NS_BC_FACE.Name = "NS_BC_FACE"
GEN_CRT1 = NS_BC_FACE.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
CRT1.Action=SelectionActionType.Add
CRT1.EntityType=SelectionType.GeoFace
CRT1.Criterion=SelectionCriterionType.LocationZ
CRT1.Operator=SelectionOperatorType.Equal
CRT1.Value=Quantity('20.00 [m]')
GEN_CRT1.Add(CRT1)
NS_BC_FACE.Activate()
NS_BC_FACE.Generate()

#Scenario 4 - Define General Joint as Translational

JOINT_GENERAL = CONNECTION_GROUP.AddJoint()
JOINT_GENERAL.ConnectionType = JointScopingType.BodyToGround
JOINT_GENERAL.Type = JointType.General
JOINT_GENERAL.TranslationZ = FixedOrFree.Free
JOINT_GENERAL.MobileLocation = NS_FIXED_FACE
# The below commands help in setting the rotational dofs
JOINT_GENERAL.Rotations = JointRotationDOFType.FreeAll
JOINT_GENERAL.Rotations = JointRotationDOFType.FixAll

#Scenario 5 - Define Mesh Settings

MESH.ElementSize = Quantity("0.25 [m]")
MESH.GenerateMesh()

#Scenario 6 - Insert Displacement BC

DISPLACEMENT_BC = STATIC_STRUCTURAL.AddDisplacement()
DISPLACEMENT_BC.Location = NS_BC_FACE
DISPLACEMENT_BC.ZComponent.Output.DiscreteValues = [Quantity("0.01 [m]")]

#Scenario 7 - Insert Results

DIRECTIONAL_DEFORMATION = STATIC_STRUCTURAL_SOLUTION.AddDirectionalDeformation()
DIRECTIONAL_DEFORMATION.NormalOrientation = NormalOrientationType.ZAxis

JOINT_PROBE_RELATIVE_DISPLACEMENT = STATIC_STRUCTURAL_SOLUTION.AddJointProbe()
JOINT_PROBE_RELATIVE_DISPLACEMENT.BoundaryConditionSelection = JOINT_GENERAL

```

```

JOINT_PROBE_RELATIVE_DISPLACEMENT.ResultType = ProbeResultType.DeformationProbe
JOINT_PROBE_RELATIVE_DISPLACEMENT.ResultSelection = ProbeDisplayFilter.ZAxis

#Scenario 8 - Solve and review the results

STATIC_STRUCTURAL_SOLUTION.Solve(True)

DIRECTIONAL_DEFORMATION_MAX = DIRECTIONAL_DEFORMATION.Maximum.Value
DIRECTIONAL_DEFORMATION_MIN = DIRECTIONAL_DEFORMATION.Minimum.Value
JOINT_PROBE_RELATIVE_DISPLACEMENT_MAX = JOINT_PROBE_RELATIVE_DISPLACEMENT.MaximumZAxis.Value
JOINT_PROBE_RELATIVE_DISPLACEMENT_MIN = JOINT_PROBE_RELATIVE_DISPLACEMENT.MinimumZAxis.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Static Structural Universal Joint Analysis

In this example, using the support files, you will insert a Static Structural analysis object into an undefined Mechanical session and execute a sequence of python journal commands that specify a Joint as Universal, define mesh and boundary and result objects and solve the analysis.

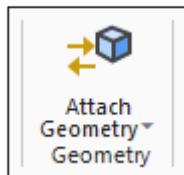
This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Mechanical_Static_Joint_002_Example.agbd
- Mechanical_Static_Joint_002_Example.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Static Structural** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the [Geometry Context tab](#). Navigate to the proper folder location and select `Mechanical_Static_Joint_002_Example.agbd`.



4. Select the **Automation** tab and select the **Scripting** option to open the **Mechanical Scripting** pane.

5. Select the **Open Script** option (from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select Mechanical_Static_Joint_002_Example.py.

6. Select the **Run Script** option (from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
=====
# ACT Mechanical API Example for General Joint as Translational
# Python Script Scenarios:
# 1. Set up the Unit Systems
# 2. Set up the Tree Object Items
# 3. Define Named Selections
# 4. Define General Joint as Translational
# 5. Define Mesh Settings
# 6. Insert Displacement BC
# 7. Insert Results
# 8. Solve and review the results

# Reference case: None. Created from scratch.
# Reference geometry: ...\\DS_Parts\\PARA_Parts\\LONGBAR.x_t
# Script is created by Rahul B Patil.
=====

#Scenario 1 - Set up the Units System

ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

#Scenario 2 - Set up the Tree Object Items

CONNECTION_GROUP = Model.Connections
MESH = Model.Mesh
STATIC_STRUCTURAL = Model.Analyses[0]
ANALYSIS_SETTINGS = STATIC_STRUCTURAL.Children[0]
STATIC_STRUCTURAL_SOLUTION = STATIC_STRUCTURAL.Solution

#Scenario 3 - Define Named Selections

NS_FIXED_FACE = DataModel.Project.Model.AddNamedSelection()
NS_FIXED_FACE.ScopingMethod=GeometryDefineByType.Worksheet
NS_FIXED_FACE.Name = "NS_FIXED_FACE"
GEN_CRT1 = NS_FIXED_FACE.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
CRT1.Action=SelectionActionType.Add
CRT1.EntityType=SelectionType.GeoFace
CRT1.Criterion=SelectionCriterionType.LocationZ
CRT1.Operator=SelectionOperatorType.Equal
CRT1.Value=Quantity('0.00 [m]')
GEN_CRT1.Add(CRT1)
NS_FIXED_FACE.Activate()
NS_FIXED_FACE.Generate()

NS_BC_FACE = DataModel.Project.Model.AddNamedSelection()
NS_BC_FACE.ScopingMethod=GeometryDefineByType.Worksheet
NS_BC_FACE.Name = "NS_BC_FACE"
GEN_CRT1 = NS_BC_FACE.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
CRT1.Action=SelectionActionType.Add
CRT1.EntityType=SelectionType.GeoFace
CRT1.Criterion=SelectionCriterionType.LocationZ
```

```
CRT1.Operator=SelectionOperatorType.Equal
CRT1.Value=Quantity('20.00 [m]')
GEN_CRT1.Add(CRT1)
NS_BC_FACE.Activate()
NS_BC_FACE.Generate()

#Scenario 4 - Define General Joint as Translational

JOINT_GENERAL = CONNECTION_GROUP.AddJoint()
JOINT_GENERAL.ConnectionType = JointScopingType.BodyToGround
JOINT_GENERAL.Type = JointType.General
JOINT_GENERAL.TranslationZ = FixedOrFree.Free
JOINT_GENERAL.MobileLocation = NS_FIXED_FACE
# The below commands help in setting the rotational dofs
JOINT_GENERAL.Rotations = JointRotationDOFType.FreeAll
JOINT_GENERAL.Rotations = JointRotationDOFType.FixAll

#Scenario 5 - Define Mesh Settings

MESH.ElementSize = Quantity("0.25 [m]")
MESH.GenerateMesh()

#Scenario 6 - Insert Displacement BC

DISPLACEMENT_BC = STATIC_STRUCTURAL.AddDisplacement()
DISPLACEMENT_BC.Location = NS_BC_FACE
DISPLACEMENT_BC.ZComponent.Output.DiscreteValues = [Quantity("0.01 [m]")]

#Scenario 7 - Insert Results

DIRECTIONAL_DEFORMATION = STATIC_STRUCTURAL_SOLUTION.AddDirectionalDeformation()
DIRECTIONAL_DEFORMATION.NormalOrientation = NormalOrientationType.ZAxis

JOINT_PROBE_RELATIVE_DISPLACEMENT = STATIC_STRUCTURAL_SOLUTION.AddJointProbe()
JOINT_PROBE_RELATIVE_DISPLACEMENT.BoundaryConditionSelection = JOINT_GENERAL
JOINT_PROBE_RELATIVE_DISPLACEMENT.ResultType = ProbeResultType.DeformationProbe
JOINT_PROBE_RELATIVE_DISPLACEMENT.ResultSelection = ProbeDisplayFilter.ZAxis

#Scenario 8 - Solve and review the results

STATIC_STRUCTURAL_SOLUTION.Solve(True)

DIRECTIONAL_DEFORMATION_MAX = DIRECTIONAL_DEFORMATION.Maximum.Value
DIRECTIONAL_DEFORMATION_MIN = DIRECTIONAL_DEFORMATION.Minimum.Value
JOINT_PROBE_RELATIVE_DISPLACEMENT_MAX = JOINT_PROBE_RELATIVE_DISPLACEMENT.MaximumZAxis.Value
JOINT_PROBE_RELATIVE_DISPLACEMENT_MIN = JOINT_PROBE_RELATIVE_DISPLACEMENT.MinimumZAxis.Value
```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Transient Structural Cylindrical Joint Analysis

In this example, using the support files, you will insert a Transient Structural analysis object into an undefined Mechanical session and execute a sequence of python journal commands that specify a Joint as Cylindrical, define mesh and boundary and result objects and solve the analysis.

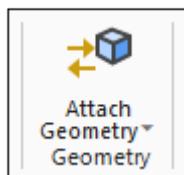
This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Cylindrical_Joint_Example.agbd
- Cylindrical_Joint_Example.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Transient Structural** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the [Geometry Context tab](#). Navigate to the proper folder location and select `Cylindrical_Joint_Example.agbd`.



4. Select the **Automation** tab and select the **Scripting** option to open the **Mechanical Scripting** pane.

5. Select the **Open Script** option (from the [Editor \(p. 4\)](#) toolbar. Navigate to the proper folder location and select `Cylindrical_Joint_Example.py`.

6. Select the **Run Script** option (from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Scenario 1 - Set up the Units System

ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardCGS

# Scenario 2 - Set up the Tree Object Items

GEOMETRY = Model.Geometry
CONNECTION_GROUP = Model.Connections
MESH = Model.Mesh
TRANSIENT_STRUCTURAL = Model.Analyses[0]
ANALYSIS_SETTINGS = TRANSIENT_STRUCTURAL.Children[1]
TRANSIENT_STRUCTURAL_SOLUTION = TRANSIENT_STRUCTURAL.Solution

# Scenario 3 - Set the Stiffness Behavior of bodies as Rigid

PART1 = GEOMETRY.Children[0].Children[0]
PART2 = GEOMETRY.Children[1].Children[0]
PART1.StiffnessBehavior = StiffnessBehavior.Rigid
```

```

PART2.StiffnessBehavior = StiffnessBehavior.Rigid

# Scenario 4 - Define Named Selections

NS_FACE_1 = DataModel.GetObjectsByName("NS_Face_1")[0]
NS_FACE_2 = DataModel.GetObjectsByName("NS_Face_2")[0]
NS_FACE_3 = DataModel.GetObjectsByName("NS_Face_3")[0]
NS_FORCE_1 = DataModel.GetObjectsByName("NS_Force_1")[0]
NS_FORCE_2 = DataModel.GetObjectsByName("NS_Force_2")[0]

# Scenario 5 - Define Cylindrical Joint and Fixed Joint

CONNECTION_GROUPActivate()
JOINT_CYLINDRICAL = CONNECTION_GROUP.AddJoint()
JOINT_CYLINDRICAL.Name = "CYLINDRICAL_JOINT"
JOINT_CYLINDRICAL.ConnectionType = JointScopingType.BodyToBody
JOINT_CYLINDRICAL.Type = JointType.Cylindrical
JOINT_CYLINDRICAL.ReferenceLocation = NS_FACE_3
JOINT_CYLINDRICAL.MobileLocation= NS_FACE_2
# User can set the value of Torsional Stiffness and Torsional Damping using below commands.
JOINT_CYLINDRICAL.TorsionalStiffness = Quantity("1e6 [dyne cm deg^-1]")
JOINT_CYLINDRICAL.TorsionalDamping= Quantity("1e6 [dyne cm s deg^-1]")

CONNECTION_GROUPActivate()
JOINT_FIXED = CONNECTION_GROUP.AddJoint()
JOINT_FIXED.Name = "FIXED_JOINT"
JOINT_FIXED.ConnectionType = JointScopingType.BodyToGround
JOINT_FIXED.Type = JointType.Fixed
JOINT_FIXED.MobileLocation= NS_FACE_1

# Scenario 6 - Define Analysis Settings

ANALYSIS_SETTINGSActivate()
ANALYSIS_SETTINGS.AutomaticTimeStepping = AutomaticTimeStepping.Off
ANALYSIS_SETTINGS.TimeStep = Quantity("1e-2 [s]")

# Scenario 7 - Insert Remote Force BC

TRANSIENT_STRUCTURALActivate()
REMOTE_FORCE_BC = TRANSIENT_STRUCTURAL.AddRemoteForce()
REMOTE_FORCE_BC.Location = NS_FORCE_2
REMOTE_FORCE_BC.DefineBy = LoadDefineBy.Components
REMOTE_FORCE_BC.YComponent.Output.DiscreteValues = [Quantity("1e11 [dyne]")]
REMOTE_FORCE_BC.ZComponent.Output.DiscreteValues = [Quantity("1e11 [dyne]")]

# Scenario 8 - Insert Results

DIRECTIONAL_DEFORMATION = TRANSIENT_STRUCTURAL_SOLUTION.AddDirectionalDeformation()
DIRECTIONAL_DEFORMATION.NormalOrientation = NormalOrientationType.ZAxis

JOINT_PROBE_RELATIVE_DISPLACEMENT = TRANSIENT_STRUCTURAL_SOLUTION.AddJointProbe()
JOINT_PROBE_RELATIVE_DISPLACEMENT.BoundaryConditionSelection = JOINT_CYLINDRICAL
JOINT_PROBE_RELATIVE_DISPLACEMENT.ResultType = ProbeResultType.DeformationProbe
JOINT_PROBE_RELATIVE_DISPLACEMENT.ResultSelection = ProbeDisplayFilter.ZAxis

JOINT_PROBE_RELATIVE_VELOCITY = TRANSIENT_STRUCTURAL_SOLUTION.AddJointProbe()
JOINT_PROBE_RELATIVE_VELOCITY.BoundaryConditionSelection = JOINT_CYLINDRICAL
JOINT_PROBE_RELATIVE_VELOCITY.ResultType = ProbeResultType.VelocityProbe
JOINT_PROBE_RELATIVE_VELOCITY.ResultSelection = ProbeDisplayFilter.ZAxis

JOINT_PROBE_RELATIVE_ACCELERATION = TRANSIENT_STRUCTURAL_SOLUTION.AddJointProbe()
JOINT_PROBE_RELATIVE_ACCELERATION.BoundaryConditionSelection = JOINT_CYLINDRICAL
JOINT_PROBE_RELATIVE_ACCELERATION.ResultType = ProbeResultType.AccelerationProbe
JOINT_PROBE_RELATIVE_ACCELERATION.ResultSelection = ProbeDisplayFilter.ZAxis

JOINT_PROBE_RELATIVE_ROTATION = TRANSIENT_STRUCTURAL_SOLUTION.AddJointProbe()

```

```

JOINT_PROBE_RELATIVE_ROTATION.BoundaryConditionSelection = JOINT_CYLINDRICAL
JOINT_PROBE_RELATIVE_ROTATION.ResultType = ProbeResultType.RotationProbe
JOINT_PROBE_RELATIVE_ROTATION.ResultSelection = ProbeDisplayFilter.ZAxis

JOINT_PROBE_RELATIVE_ANGULAR_VELOCITY = TRANSIENT_STRUCTURAL SOLUTION.AddJointProbe()
JOINT_PROBE_RELATIVE_ANGULAR_VELOCITY.BoundaryConditionSelection = JOINT_CYLINDRICAL
JOINT_PROBE_RELATIVE_ANGULAR_VELOCITY.ResultType = ProbeResultType.AngularVelocityProbe
JOINT_PROBE_RELATIVE_ANGULAR_VELOCITY.ResultSelection = ProbeDisplayFilter.ZAxis

JOINT_PROBE_RELATIVE_ANGULAR_ACCELERATION = TRANSIENT_STRUCTURAL SOLUTION.AddJointProbe()
JOINT_PROBE_RELATIVE_ANGULAR_ACCELERATION.BoundaryConditionSelection = JOINT_CYLINDRICAL
JOINT_PROBE_RELATIVE_ANGULAR_ACCELERATION.ResultType = ProbeResultType.AngularAccelerationProbe
JOINT_PROBE_RELATIVE_ANGULAR_ACCELERATION.ResultSelection = ProbeDisplayFilter.ZAxis

JOINT_PROBE_TOTAL_FORCE = TRANSIENT_STRUCTURAL SOLUTION.AddJointProbe()
JOINT_PROBE_TOTAL_FORCE.BoundaryConditionSelection = JOINT_CYLINDRICAL
JOINT_PROBE_TOTAL_FORCE.ResultType = ProbeResultType.ForceReaction
JOINT_PROBE_TOTAL_FORCE.ResultSelection = ProbeDisplayFilter.YAxis

JOINT_PROBE_TOTAL_MOMENT = TRANSIENT_STRUCTURAL SOLUTION.AddJointProbe()
JOINT_PROBE_TOTAL_MOMENT.BoundaryConditionSelection = JOINT_CYLINDRICAL
JOINT_PROBE_TOTAL_MOMENT.ResultType = ProbeResultType.MomentReaction
JOINT_PROBE_TOTAL_MOMENT.ResultSelection = ProbeDisplayFilter.YAxis

# Scenario 9 - Solve and review the results

TRANSIENT_STRUCTURAL SOLUTION.Solve(True)

JOINT_PROBE_RELATIVE_DISPLACEMENT_VAL = JOINT_PROBE_RELATIVE_DISPLACEMENT.ZAxis.Value
JOINT_PROBE_RELATIVE_VELOCITY_VAL = JOINT_PROBE_RELATIVE_VELOCITY.ZAxis.Value
JOINT_PROBE_RELATIVE_ACCELERATION_VAL = JOINT_PROBE_RELATIVE_ACCELERATION.ZAxis.Value
JOINT_PROBE_RELATIVE_ROTATION_VAL = JOINT_PROBE_RELATIVE_ROTATION.ZAxis.Value
JOINT_PROBE_RELATIVE_ANGULAR_VELOCITY_VAL = JOINT_PROBE_RELATIVE_ANGULAR_VELOCITY.ZAxis.Value
JOINT_PROBE_RELATIVE_ANGULAR_ACCELERATION_VAL = JOINT_PROBE_RELATIVE_ANGULAR_ACCELERATION.ZAxis.Value
JOINT_PROBE_TOTAL_FORCE_VAL = JOINT_PROBE_TOTAL_FORCE.YAxis.Value
JOINT_PROBE_TOTAL_MOMENT_VAL = JOINT_PROBE_TOTAL_MOMENT.YAxis.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Symmetric Symmetry Analysis

In this example, using the support files, you will perform a Static Structural analysis using Symmetric Symmetry and execute a sequence of python journal commands that will define and solve the analysis.

This example begins in the Mechanical application. It requires you to download the following files.

- Symmetric_Symmetry.agdb
- Symmetric_Symmetry.xml
- Symmetric_Symmetry.py

These files are available [here](#).

Procedure

1. Open Workbench and insert a **Static Structural** system into the Project Schematic.

2. Double-click the **Engineering Data** cell to open the workspace.
3. Select **File > Import Engineering Data**, navigate to the proper folder location and select Symmetric_Symmetry.xml.
4. Return to the **Project** tab.
5. Right-click the **Geometry** cell and select **Properties**.
6. Enable the **Named Selections** check box under the **Basic Geometry Options** category.
7. Right-click the **Geometry** cell and select **Import Geometry > Browse** and then navigate to the proper folder location and select Symmetric_Symmetry.agdb.
8. Open Mechanical: right-click the **Model** cell and select **Edit**.
9. Select the **Automation** tab and select the **Scripting** option to open the **Mechanical Scripting** pane.



10. Select the **Open Script** option (the icon with a folder and a red arrow) from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select Symmetric_Symmetry.py.



11. Select the **Run Script** option (the icon with a play button) from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
#Scenario 1: Store main Tree Object items
MODEL = Model
GEOM = MODEL.Geometry
COORDINATE_SYSTEMS = Model.CoordinateSystems
MESH = Model.Mesh

PARTS = GEOM.GetChildren(DataModelObjectCategory.Part, False)
for part in PARTS:
    bodies = part.GetChildren(DataModelObjectCategory.Body, False)
    for body in bodies:
        if body.Name == "Part 1":
            PART1 = body

STAT_STRUC = DataModel.Project.Model.Analyses[0]
ANALYSIS_SETTINGS = STAT_STRUC.AnalysisSettings
STAT_STRUC SOLUTION = STAT_STRUC.Solution

#Scenario 2: Set Display Unit System
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

#Scenario 3: Insert Worksheet based Named Selections for applying Symmetric Symmetry and to setup Static Structural
# Create Named Selection for applying Symmetric Symmetry
FIRSTNS = DataModel.Project.Model.AddNamedSelection()
FIRSTNS.ScopingMethod=GeometryDefineByType.Worksheet
FIRSTNS.Name = "SELECTION2"
GEN_CRT1 = FIRSTNS.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
CRT1.Action=SelectionActionType.Add
CRT1.EntityType=SelectionType.GeoFace
CRT1.Criterion=SelectionCriterionType.LocationY
CRT1.Operator=SelectionOperatorType.Equal
CRT1.Value=Quantity('0 [m]')
```

```

GEN_CRT1.Add(CRT1)
FIRSTNS.Activate()
FIRSTNS.Generate()

# Create Named Selection for applying Fixed Support
NAMED_SELCTIONS = DataModel.Project.Model.NamedSelections
FIRSTNS.Duplicate()
SECONDNS=NAMED_SELCTIONS.Children[1]
SECONDNS.Name = "SELECTION3"
GEN_CRT2 = SECONDNS.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
GEN_CRT2[0].Criterion=SelectionCriterionType.LocationX
GEN_CRT2[0].Value=Quantity('-7.5 [m]')
SECONDNS.Activate()
SECONDNS.Generate()

# Create Named Selection for applying Pressure
SECONDNS.Duplicate()
THIRDNS=NAMED_SELCTIONS.Children[2]
THIRDNS.Name = "SELECTION4"
GEN_CRT3 = THIRDNS.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
GEN_CRT3[0].Value=Quantity('7.5 [m]')
THIRDNS.Activate()
THIRDNS.Generate()

#Scenario 4: Assign proper material to body")
PART1Activate()
PART1.Assignment="T5"

#Scenario 5: Insert Coordinate System for applying Symmetric Symmetry
COORDINATE_SYSTEM = COORDINATE_SYSTEMS.AddCoordinateSystem()
COORDINATE_SYSTEM.OriginZ = Quantity("0.1 [m]")
COORDINATE_SYSTEM.AddTransformation(TransformationType.Offset,CoordinateSystemAxisType.PositiveZAxis)
COORDINATE_SYSTEM.SetTransformationValue(1,0.4)
COORDINATE_SYSTEM.AddTransformation(TransformationType.Rotation,CoordinateSystemAxisType.PositiveXAxis)
COORDINATE_SYSTEM.SetTransformationValue(2,90)

#Scenario 6: Insert Symmetric Symmetry using local Coordinate System
SYMMETRY=MODEL.AddSymmetry()
# Insert Symmetric Symmetry using Pre-selection
SELECTION=ExtAPI.SelectionManager.AddSelection(FIRSTNS)
SELECTION2=ExtAPI.SelectionManager.CurrentSelection
SYMMETRY_REGION=SYMMETRY.AddSymmetryRegion()
SYMMETRY_REGION.Type=SymmetryRegionType.Symmetric
SYMMETRY_REGION.CoordinateSystem=COORDINATE_SYSTEM
SYMMETRY_REGION.SymmetryNormal=SymmetryNormalType.ZAxis
# Clear Selection for inserting new objects with default scoping
CLRSEL = ExtAPI.SelectionManager.ClearSelection()

#Scenario 7: Define global mesh size and generate mesh
MESHActivate()
MESH.ElementSize = Quantity('0.5 [m]')
MESH.GenerateMesh()

#Scenario 8: Define load and boundary conditions in Static Structural analysis
# Add Fixed Support
FIX_SUPPORT = STAT_STRUC.AddFixedSupport()
FIX_SUPPORT.Location = SECONDNS

# Add Pressure
PRESSURE = STAT_STRUC.AddPressure()
PRESSURE.Location = THIRDNS
PRESSURE.AppliedBy=LoadAppliedBy.SurfaceEffect
PRESSURE.Magnitude.Output.SetDiscreteValue(0,Quantity("-100 [Pa]"))

#Scenario 9: Add results in Static Structural analysis
STAT_STRUC SOLUTION.Activate()
TOTAL_DEFORMATION = STAT_STRUC SOLUTION.AddTotalDeformation()

```

```

NORMAL_STRESS = STAT_STRUC_SOLUTION.AddNormalStress()

#Scenario 10: Solve and review Results
STAT_STRUC_SOLUTION.Activate()
STAT_STRUC.Solve(True)

#Total Deformation Result
TOTAL_DEF_MIN = TOTAL_DEFORMATION.Minimum.Value
TOTAL_DEF_MAX = TOTAL_DEFORMATION.Maximum.Value

#Normal Stress Result
NORMAL_STRESS_MIN = NORMAL_STRESS.Minimum.Value
NORMAL_STRESS_MAX = NORMAL_STRESS.Maximum.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Linear Periodic Symmetry Analysis

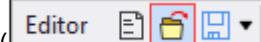
In this example, using the support files, you will perform a Static Structural analysis using Linear Periodic Symmetry and execute a sequence of python journal commands that will define and solve the analysis.

This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Linear_Periodic_Symmetry.agdb
- Linear_Periodic_Symmetry.py

These files are available [here](#).

Procedure

1. Open Workbench and insert a **Static Structural** system into the Project Schematic.
2. Right-click the **Geometry** cell and select **Properties**.
3. Enable the **Named Selections** check box under the **Basic Geometry Options** category.
4. Right-click the **Geometry** cell and select **Import Geometry > Browse** and then navigate to the proper folder location and select `Linear_Periodic_Symmetry.agdb`.
5. Open Mechanical: right-click the **Model** cell and select **Edit**.
6. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.
7. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select `Linear_Periodic_Symmetry.py`.
8. Select the **Run Script** option () from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
#Scenario 1: Store main Tree Object items
MODEL = Model
GEOM = MODEL.Geometry
COORDINATE_SYSTEMS = Model.CoordinateSystems
MESH = Model.Mesh
NAMED_SELECTIONS = Model.NamedSelections

PARTS = GEOM.GetChildren(DataModelObjectCategory.Part, False)
for part in PARTS:
    bodies = part.GetChildren(DataModelObjectCategory.Body, False)
    for body in bodies:
        if body.Name == "Solid":
            SOLID = body

STAT_STRUC = DataModel.Project.Model.Analyses[0]
ANALYSIS_SETTINGS = STAT_STRUC.AnalysisSettings
STAT_STRUC_SOLUTION = STAT_STRUC.Solution

#Scenario 2: Set Display Unit System
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardCGS

#Scenario 3: Store Named Selections for applying Symmetry and to setup Static Structural analysis
tot_child_NS_GRP = NAMED_SELECTIONS.Children.Count
for i in range(0, tot_child_NS_GRP, 1):
    ns = NAMED_SELECTIONS.Children[i].Name
    if(ns=='Left_Face'):
        LOW_FACE = NAMED_SELECTIONS.Children[i]
    if(ns=='Right_Face'):
        HIGH_FACE = NAMED_SELECTIONS.Children[i]
    if(ns=='Base_Face'):
        LOAD = NAMED_SELECTIONS.Children[i]
    if(ns=='Top_Face'):
        FIXED = NAMED_SELECTIONS.Children[i]
    if(ns=='Front_Face'):
        FRONT_FACE = NAMED_SELECTIONS.Children[i]
    if(ns=='Back_Face'):
        BACK_FACE = NAMED_SELECTIONS.Children[i]
    if(ns=='Curved_Edges'):
        CURVED_EDGES = NAMED_SELECTIONS.Children[i]
    if(ns=='Straight_Edges'):
        STRAIGHT_EDGES = NAMED_SELECTIONS.Children[i]
    if(ns=='Short_Edges'):
        SHORT_EDGES = NAMED_SELECTIONS.Children[i]
    if(ns=='Mapping_Faces'):
        MAPPING_FACES = NAMED_SELECTIONS.Children[i]
    if(ns=='Vertex1'):
        VERETX1 = NAMED_SELECTIONS.Children[i]
    if(ns=='Vertex2'):
        VERETX2 = NAMED_SELECTIONS.Children[i]
    if(ns=='Vertex3'):
        VERETX3 = NAMED_SELECTIONS.Children[i]
    if(ns=='Vertex4'):
        VERETX4 = NAMED_SELECTIONS.Children[i]
    if(ns=='Vertex5'):
        VERETX5 = NAMED_SELECTIONS.Children[i]
    if(ns=='Vertex6'):
        VERETX6 = NAMED_SELECTIONS.Children[i]
    if(ns=='Vertex7'):
        VERETX7 = NAMED_SELECTIONS.Children[i]
    if(ns=='Vertex8'):
        VERETX8 = NAMED_SELECTIONS.Children[i]

#Scenario 4: Insert Coordinate System for applying Linear Periodic Symmetry
COORDINATE_SYSTEM = COORDINATE_SYSTEMS.AddCoordinateSystem()
COORDINATE_SYSTEM.OriginX = Quantity("8 [cm]")
COORDINATE_SYSTEM.OriginY = Quantity("4 [cm]")
```

```

COORDINATE_SYSTEM.OriginZ = Quantity("0 [cm]")
COORDINATE_SYSTEM.AddTransformation(TransformationType.Rotation,CoordinateSystemAxisType.PositiveYAxis)
COORDINATE_SYSTEM.SetTransformationValue(1,90)

#Scenario 5: Insert Linear Periodic Symmetry using local Coordinate System
SYMMETRY=MODEL.AddSymmetry()
# Insert Linear Periodic Symmetry
SYMMETRY_REGION=SYMMETRY.AddLinearPeriodicRegion()
SYMMETRY_REGION.Behavior=SymmetryBehavior.Free
SYMMETRY_REGION.CoordinateSystem=COORDINATE_SYSTEM
SYMMETRY_REGION.LinearShift=Quantity("8 [cm]")
SYMMETRY_REGION.PeriodicityDirection=PeriodicityDirectionType.ZAxis
# Select Low and High Boundary with proper Coordinate System
SYMMETRY_REGION.LowBoundaryLocation=HIGH_FACE
SYMMETRY_REGION.HighBoundaryLocation=LOW_FACE
SYMMETRY_REGION.FlipHighLow()

#Scenario 6: Define mesh controls and generate mesh
MESH.Activate()
MESH.ElementOrder=ElementOrder.Quadratic
# Error Limits Standard Mechanical
MESH.ShapeChecking=False

MESH_SIZE01 = MESH.AddSizing()
MESH_SIZE01.Location = CURVED_EDGES
MESH_SIZE01.Type = SizingType.NumberOfDivisions
MESH_SIZE01.NumberOfDivisions = 10
MESH_SIZE01.Behavior=SizingBehavior.Hard

MESH_SIZE02 = MESH_SIZE01.Duplicate()
MESH_SIZE02.Location = STRAIGHT_EDGES
MESH_SIZE02.NumberOfDivisions = 10

MESH_SIZE03 = MESH_SIZE02.Duplicate()
MESH_SIZE03.Location = SHORT_EDGES
MESH_SIZE03.NumberOfDivisions = 3

FACE_MESHING = MESH.AddFaceMeshing()
FACE_MESHING.Location = MAPPING_FACES

MESH.GenerateMesh()

#Scenario 7: Define load and boundary conditions in Static Structural analysis
# Add Fixed Support
FIXED_SUPPORT = STAT_STRUC.AddFixedSupport()
FIXED_SUPPORT.Location = FIXED

# Add Pressure
PRESSURE = STAT_STRUC.AddPressure()
PRESSURE.Location = LOAD
PRESSURE.AppliedBy=LoadAppliedBy.SurfaceEffect
PRESSURE.DefineBy=LoadDefineBy.Components
PRESSURE.XComponent.Output.DiscreteValues = [Quantity("1e9 [dyne cm^-1 cm^-1]")]

#Scenario 8: Add results in Static Structural analysis
STAT_STRUC SOLUTION.Activate()
# Insert Deformation results
TOTAL_DEFORMATION1 = STAT_STRUC SOLUTION.AddTotalDeformation()
TOTAL_DEFORMATION2 = STAT_STRUC SOLUTION.AddTotalDeformation()
TOTAL_DEFORMATION2.Location=VERETX3
TOTAL_DEFORMATION3 = STAT_STRUC SOLUTION.AddTotalDeformation()
TOTAL_DEFORMATION3.Location=VERETX7
TOTAL_DEFORMATION4 = STAT_STRUC SOLUTION.AddTotalDeformation()
TOTAL_DEFORMATION4.Location=VERETX4
TOTAL_DEFORMATION5 = STAT_STRUC SOLUTION.AddTotalDeformation()
TOTAL_DEFORMATION5.Location=VERETX8
DIRECTIONAL_DEFORMATION = STAT_STRUC SOLUTION.AddDirectionalDeformation()
# Insert Equivalent Stress result
EQUIVALENT_STRESS = STAT_STRUC SOLUTION.AddEquivalentStress()
#Insert Force Reaction Probe scoped to Fixed Support
FORCEREACTION_PROBE = STAT_STRUC.Solution.AddForceReaction()

```

```

FORCE_REACTION_PROBE.LocationMethod = LocationDefinitionMethod.BoundaryCondition
FORCE_REACTION_PROBE.BoundaryConditionSelection = FIXED_SUPPORT

#Scenario 9: Solve and review Results
STAT_STRUC_SOLUTION.Activate()
STAT_STRUC.Solve(True)

#Deformation Result
TOTAL_DEF1_MIN = TOTAL_DEFORMATION1.Minimum.Value
TOTAL_DEF1_MAX = TOTAL_DEFORMATION1.Maximum.Value

TOTAL_DEF2_MIN = TOTAL_DEFORMATION2.Minimum.Value
TOTAL_DEF2_MAX = TOTAL_DEFORMATION2.Maximum.Value

TOTAL_DEF3_MIN = TOTAL_DEFORMATION3.Minimum.Value
TOTAL_DEF3_MAX = TOTAL_DEFORMATION3.Maximum.Value

TOTAL_DEF4_MIN = TOTAL_DEFORMATION4.Minimum.Value
TOTAL_DEF4_MAX = TOTAL_DEFORMATION4.Maximum.Value

TOTAL_DEF5_MIN = TOTAL_DEFORMATION5.Minimum.Value
TOTAL_DEF5_MAX = TOTAL_DEFORMATION5.Maximum.Value

#Equivalent Stress Result
EQV_STRESS_MIN = EQUIVALENT_STRESS.Minimum.Value
EQV_STRESS_MAX = EQUIVALENT_STRESS.Maximum.Value

FRC_REAC_PROBE_MAX_Y=FORCE_REACTION_PROBE.MaximumXAxis.Value
FRC_REAC_PROBE_TOTAL=FORCE_REACTION_PROBE.MaximumTotal.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Cyclic Symmetry Analysis

In this example, using the support files, you will perform a Thermal Stress analysis using Cyclic Symmetry or Pre-Meshed Cyclic Region and execute a sequence of python journal commands that will define and solve the analysis.

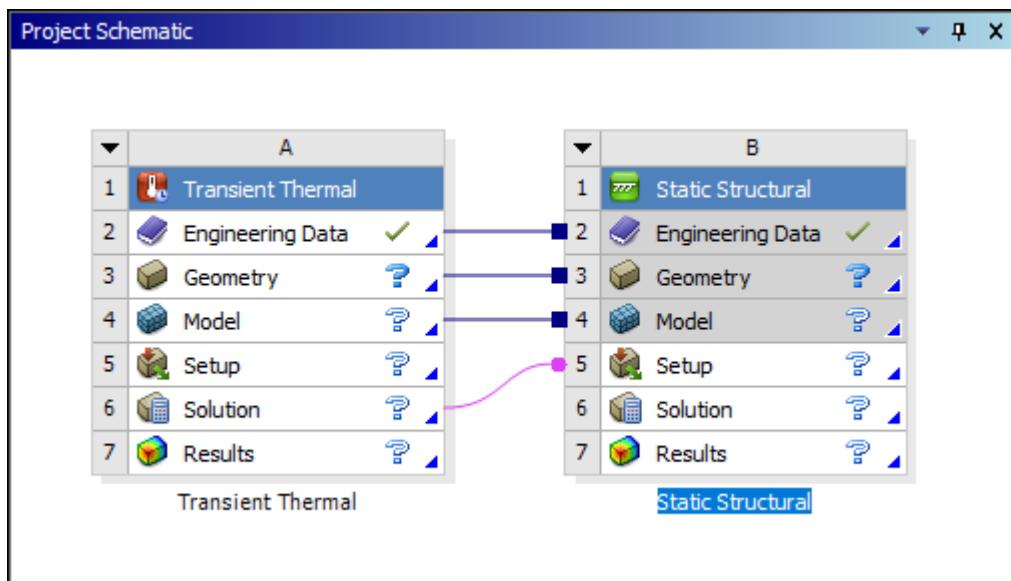
This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Cyclic_Symmetry.agdb
- Cyclic_Symmetry.py

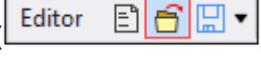
These files are available [here](#).

Procedure

1. Open Workbench and insert a **Transient Thermal** system into the Project Schematic.
2. Drag and drop an **Static Structural** system onto the **Solution** cell of the thermal analysis so that the systems are linked as shown.



3. Right-click the **Geometry** cell of the thermal system and select **Properties**.
4. Enable the **Named Selections** check box under the **Basic Geometry Options** category.
5. Right-click the **Geometry** cell of the thermal system and select **Import Geometry > Browse** and then navigate to the proper folder location and select *Cyclic_Symmetry.agdb*.
6. Open Mechanical: right-click the **Model** cell and select **Edit**.
7. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.

8. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select *Cyclic_Symmetry.py*.
9. Select the **Run Script** option () from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
#Scenario 1: Store main Tree Object items
MODEL = Model
GEOM = MODEL.Geometry
COORDINATE_SYSTEMS = Model.CoordinateSystems
MESH = Model.Mesh
NAMED_SELECTIONS = Model.NamedSelections

PARTS = GEOM.GetChildren(DataModelObjectCategory.Part, False)
for part in PARTS:
    bodies = part.GetChildren(DataModelObjectCategory.Body, False)
    for body in bodies:
        if body.Name == "Solid":
            SOLID_BODY = body
        if body.Name == "SurfaceBody":
            SHELL_BODY = body

TRANS_THERM = DataModel.Project.Model.Analyses[0]
ANALYSIS_SETTINGS1 = TRANS_THERM.AnalysisSettings
```

```

TRANS_THERM SOLUTION = TRANS_THERM.Solution
STAT_STRUC = DataModel.Project.Model.Analyses[1]
ANALYSIS_SETTINGS2 = STAT_STRUC.AnalysisSettings
STAT_STRUC SOLUTION = STAT_STRUC.Solution

#Scenario 2: Set Display Unit System
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardNMM

#Scenario 3: Set isometric view and zoom to fit
cam = Graphics.Camera
cam.SetSpecificViewOrientation(ViewOrientationType.Iso)
cam.SetFit()

#Scenario 4: Store Named Selections for applying Symmetry and to setup Thermal Stress analysis
tot_child_NS_GRP = NAMED_SELECTIONS.Children.Count
for i in range(0, tot_child_NS_GRP, 1):
    ns = NAMED_SELECTIONS.Children[i].Name
    if(ns=='Low_Face'):
        LOW_FACE = NAMED_SELECTIONS.Children[i]
    if(ns=='High_Face'):
        HIGH_FACE = NAMED_SELECTIONS.Children[i]
    if(ns=='Pressure_Face'):
        LOAD_FACE = NAMED_SELECTIONS.Children[i]
    if(ns=='Fixed_Face'):
        FIXED_FACE = NAMED_SELECTIONS.Children[i]
    if(ns=='Front_Face'):
        FRONT_FACE = NAMED_SELECTIONS.Children[i]
    if(ns=='Back_Face'):
        BACK_FACE = NAMED_SELECTIONS.Children[i]
    if(ns=='Edges1'):
        EDGES1 = NAMED_SELECTIONS.Children[i]
    if(ns=='Edges2'):
        EDGES2 = NAMED_SELECTIONS.Children[i]
    if(ns=='Edges3'):
        EDGES3 = NAMED_SELECTIONS.Children[i]
    if(ns=='Low_Edge'):
        LOW_EDGE = NAMED_SELECTIONS.Children[i]
    if(ns=='High_Edge'):
        HIGH_EDGE = NAMED_SELECTIONS.Children[i]
    if(ns=='Shell_Face'):
        SHELL_FACE = NAMED_SELECTIONS.Children[i]
    if(ns=='Solid_Body'):
        SOLID_BODY = NAMED_SELECTIONS.Children[i]
    if(ns=='Shell_Body'):
        SHELL_BODY = NAMED_SELECTIONS.Children[i]

#Scenario 5: Insert Coordinate System for applying Cyclic Symmetry
COORDINATE_SYSTEM = COORDINATE_SYSTEMS.AddCoordinateSystem()
COORDINATE_SYSTEM.OriginX = Quantity("0 [mm]")
COORDINATE_SYSTEM.OriginY = Quantity("0 [mm]")
COORDINATE_SYSTEM.OriginZ = Quantity("0 [mm]")
COORDINATE_SYSTEM.PrimaryAxis=CoordinateSystemAxisType.PositiveZAxis
COORDINATE_SYSTEM.CoordinateSystemType=CoordinateSystemTypeEnum.Cylindrical

#Scenario 6: Insert Cyclic Symmetry using local Coordinate System
SYMMETRY=MODEL.AddSymmetry()
# Insert Cyclic Symmetry scoped to Solid faces
SYMMETRY_REGION1=SYMMETRY.AddCyclicRegion()
SYMMETRY_REGION1.CoordinateSystem=COORDINATE_SYSTEM
# Select Low and High Boundary with proper Coordinate System
SYMMETRY_REGION1.LowBoundaryLocation=LOW_FACE
SYMMETRY_REGION1.HighBoundaryLocation=HIGH_FACE

#Scenario 7: Define mesh controls and generate mesh
MESH.Activate()
# Error Limits Standard Mechanical
MESH.ShapeChecking=False

FACE_MESHING = MESH.AddFaceMeshing()
FACE_MESHING.Location = SHELL_FACE
FACE_MESHING.Suppresses=True

```

```

MESH_SIZE01 = MESH.AddSizing()
MESH_SIZE01.Location = EDGES1
MESH_SIZE01.Type = SizingType.NumberOfDivisions
MESH_SIZE01.NumberOfDivisions = 9
MESH_SIZE01.Behavior=SizingBehavior.Hard
MESH_SIZE01.Suppresssed=True

MATCH_CONTROL01=MESH.AddMatchControl()
MATCH_CONTROL01.LowGeometrySelection=LOW_FACE
MATCH_CONTROL01.HighGeometrySelection=HIGH_FACE
MATCH_CONTROL01.HighCoordinateSystem=COORDINATE_SYSTEM
MATCH_CONTROL01.Suppresssed=True

MATCH_CONTROL02=MESH.AddMatchControl()
MATCH_CONTROL02.LowGeometrySelection=LOW_EDGE
MATCH_CONTROL02.HighGeometrySelection=HIGH_EDGE
MATCH_CONTROL02.HighCoordinateSystem=COORDINATE_SYSTEM
MATCH_CONTROL02.Suppresssed=True

MESH.GenerateMesh()

#Scenario 8: Setup Transient Thermal analysis
ANALYSIS_SETTINGS1.SetStepEndTime(1, Quantity("10 [sec]"))

TEMPERATURE01 = TRANS_THERM.AddTemperature()
TEMPERATURE01.Location = FIXED_FACE
TEMPERATURE01.Magnitude.Output.DiscreteValues=[Quantity('5[C]')]

TEMPERATURE02 = TRANS_THERM.AddTemperature()
TEMPERATURE02.Location = SHELL_FACE
TEMPERATURE02.Magnitude.Output.DiscreteValues=[Quantity('30[C]')]

#Scenario 9: Define load and boundary conditions in Thermal Stress analysis
STAT_STRUC.Activate()
TOT_CHILD_STAT_STRUC = STAT_STRUC.Children.Count
for i in range(0, TOT_CHILD_STAT_STRUC, 1):
    ns = STAT_STRUC.Children[i].Name
    if(ns=='Imported Load (A6) '):
        IMPORTED_LOAD_GRP = STAT_STRUC.Children[i]

IMPORTED_BODY_TEMP01=IMPORTED_LOAD_GRP.Children[0]
IMPORTED_BODY_TEMP01.Location=SOLID_BODY

IMPORTED_BODY_TEMP02=IMPORTED_LOAD_GRP.AddImportedBodyTemperature()
IMPORTED_BODY_TEMP02.Location=SHELL_BODY

# Add Fixed Support
FIXED_SUPPORT = STAT_STRUC.AddFixedSupport()
FIXED_SUPPORT.Location = FIXED_FACE

# Add Pressure
PRESSURE = STAT_STRUC.AddPressure()
PRESSURE.Location = LOAD_FACE
PRESSURE.AppliedBy=LoadAppliedBy.SurfaceEffect
PRESSURE.DefineBy=LoadDefineBy.Components
PRESSURE.XComponent.Output.DiscreteValues = [Quantity("2 [MPa]")]

#Scenario 10: Add results in Transient Thermal analysis
TRANS_THERM SOLUTION.Activate()

TEMPERATURE01_RST = TRANS_THERM SOLUTION.AddTemperature()

#Scenario 11: Add results in Thermal Stress analysis
STAT_STRUC SOLUTION.Activate()
# Insert Deformation results
TOTAL_DEFORMATION = STAT_STRUC SOLUTION.AddTotalDeformation()
# Insert Equivalent Stress result
EQUIVALENT_STRESS = STAT_STRUC SOLUTION.AddEquivalentStress()

#Scenario 12: Solve and review Results using Cyclic Symmetry

```

```

TRANS_THERM SOLUTION.Activate()
TRANS_THERM SOLUTION.Solve(True)

#Temperature Result
TEMPERATURE_RST_MIN = TEMPERATURE01_RST.Minimum.Value
TEMPERATURE_RST_MAX = TEMPERATURE01_RST.Maximum.Value

STAT_STRUC SOLUTION.Activate()
STAT_STRUC SOLUTION.Solve(True)

#Deformation Result
TOTAL_DEF_MIN = TOTAL_DEFORMATION.Minimum.Value
TOTAL_DEF_MAX = TOTAL_DEFORMATION.Maximum.Value

#Equivalent Stress Result
EQV_STRESS_MIN = EQUIVALENT_STRESS.Minimum.Value
EQV_STRESS_MAX = EQUIVALENT_STRESS.Maximum.Value

#Scenario 13: Suppress Cyclic Symmetry and insert Pre-Meshed Cyclic Region using local Coordinate System
SYMMETRY_REGION1.Suppresses=True

SYMMETRY_REGION2=SYMMETRY.AddPreMeshedCyclicRegion()
SYMMETRY_REGION2.CoordinateSystem=COORDINATE_SYSTEM
# Select Low and High Boundary with proper Coordinate System
SYMMETRY_REGION2.LowBoundaryLocation=LOW_FACE
SYMMETRY_REGION2.HighBoundaryLocation=HIGH_FACE
SYMMETRY_REGION2.NumberOfSectors=4

#Scenario 14: Activate few mesh controls to get mapping mesh and generate mesh
MESHActivate()
#Activate few mesh controls to get mapped meshed on low and high faces
FACE_MESHING.Suppresses=False
MESH_SIZE01.Suppresses=False
MATCH_CONTROL01.Suppresses=False
MATCH_CONTROL02.Suppresses=False

MESH.GenerateMesh()

#Scenario 15: Solve and review Results using Pre-Meshed Cyclic Region
TRANS_THERM SOLUTION.Activate()
TRANS_THERM SOLUTION.Solve(True)

#Temperature Result
TEMPERATURE2_RST_MIN = TEMPERATURE01_RST.Minimum.Value
TEMPERATURE2_RST_MAX = TEMPERATURE01_RST.Maximum.Value

STAT_STRUC SOLUTION.Activate()
STAT_STRUC SOLUTION.Solve(True)

#Deformation Result
TOTAL_DEF2_MIN = TOTAL_DEFORMATION.Minimum.Value
TOTAL_DEF2_MAX = TOTAL_DEFORMATION.Maximum.Value

#Equivalent Stress Result
EQV_STRESS2_MIN = EQUIVALENT_STRESS.Minimum.Value
EQV_STRESS2_MAX = EQUIVALENT_STRESS.Maximum.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Steady-State Thermal Analysis

In this example, using the support files, you will insert a Steady-State Thermal analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will define and solve the analysis.

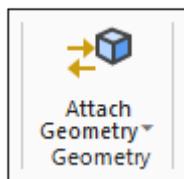
This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Mechanical_Steady_State_Thermal_Example_001.agdb
- Mechanical_Steady_State_Thermal_Example_001.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through **Start Menu**.
2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Steady-State Thermal** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the **Geometry Context tab**. Navigate to the proper folder location and select **Mechanical_Steady_State_Thermal_Example_001.agdb**.



4. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.

5. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select **Mechanical_Steady_State_Thermal_Example_001.py**.

6. Select the **Run Script** option () from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Scenario1 - Set up Unit System: Set up the Unit Systems to Metric (mm, kg, N, s, mV, mA),
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardNMM

# Scenario 2 - Set up the Tree Object Items: Define the parts, connections, mesh, Steady-State Thermal system, the
PART1 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Part 1'][0]
PART2 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Part 2'][0]
```

```

PART3 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Part 3'][0]
MESH = Model.Mesh
STEADY_STATE_THERMAL = DataModel.AnalysisByName("Steady-State Thermal")
ANALYSIS_SETTINGS = STEADY_STATE_THERMAL.AnalysisSettings
SOLUTION = STEADY_STATE_THERMAL.Solution

# Scenario 3 - Define Named Selections: Define the Named selections that will be used to define the Mesh Controls, ...
NS_CONVECTION_FACE = DataModel.GetObjectsByName("NS_convection_face")[0]
NS_HG = DataModel.GetObjectsByName("NS_heat_generation")[0]
NS_RESULTS = DataModel.GetObjectsByName("NS_results")[0]
NS_ALL_BODIES = DataModel.GetObjectsByName("NS_all_bodies")[0]

# Scenario 4 - Define Mesh Settings: Set the mesh resolution to 3. Define mesh control with sweep method, mesh algo...
MESH.Resolution = 3
MESH_METHOD = MESH.AddAutomaticMethod()
MESH_METHOD.Location = NS_ALL_BODIES
MESH_METHOD.Method = MethodType.Sweep
MESH_METHOD.Algorithm = MeshMethodAlgorithm.Axisymmetric
MESH_METHOD.MeshInCenter = 0
MESH_METHOD.SweepNumberDivisions = 24
MESH.GenerateMesh()

# Scenario 5 - Define Analysis Settings: Set the auto time stepping to on. Set the initial, minimum and maximum sub...
ANALYSIS_SETTINGS.SetAutomaticTimeStepping(1, AutomaticTimeStepping.On)
ANALYSIS_SETTINGS.DefineBy = TimeStepDefineByType.Substeps
ANALYSIS_SETTINGS.InitialSubsteps = 20
ANALYSIS_SETTINGS.MinimumSubsteps = 15
ANALYSIS_SETTINGS.MaximumSubsteps = 50

# Scenario 6 - Insert Thermal loads: Insert and define Convection and Internal Heat Generation loads
CONVECTION = STEADY_STATE_THERMAL.AddConvection()
CONVECTION.Location = NS_CONVECTION_FACE
CONVECTION.FilmCoefficient.Output.DiscreteValues=[Quantity('3.2e-4[W mm^-1 mm^-1 C^-1]'), Quantity('1.04e-4[W mm^-1 mm^-1 C^-1]')]
CONVECTION.AmbientTemperature.Output.DiscreteValues=[Quantity('50 [C]'), Quantity('200 [C]')]

INTERNAL_HEAT_GENERATION = STEADY_STATE_THERMAL.AddInternalHeatGeneration()
INTERNAL_HEAT_GENERATION.Location = NS_ALL_BODIES
INTERNAL_HEAT_GENERATION.Magnitude.Output.SetDiscreteValue(0, Quantity(0.001592, "W mm^-1 mm^-1 mm^-1"))

# Scenario 7 - Insert Results: Insert a Temperature, Total Heat Flux and Reaction Probe results.
TEMPERATURE01 = SOLUTION.AddTemperature()
TEMPERATURE01.Location = NS_CONVECTION_FACE
TOTAL_HEAT_FLUX01 = SOLUTION.AddTotalHeatFlux()
TOTAL_HEAT_FLUX01.Location = NS_CONVECTION_FACE
TEMPERATURE02 = SOLUTION.AddTemperature()
TEMPERATURE02.Location = NS_RESULTS
TOTAL_HEAT_FLUX02 = SOLUTION.AddTotalHeatFlux()
TOTAL_HEAT_FLUX02.Location = NS_RESULTS
REACTION_PROBE = SOLUTION.AddReactionProbe()
REACTION_PROBE.BoundaryConditionSelection = CONVECTION

# Scenario 8 - Solve and Define the Results: Solve the system and set the results variables. Note the Scripting wind...
SOLUTION.Solve(True)

TEMPERATURE01_MAX = TEMPERATURE01.Maximum.Value
TEMPERATURE02_MAX = TEMPERATURE02.Maximum.Value
TOTAL_HEAT_FLUX01_MAX = TOTAL_HEAT_FLUX01.Maximum.Value
TOTAL_HEAT_FLUX02_MAX = TOTAL_HEAT_FLUX02.Maximum.Value
REACTION_PROBE = REACTION_PROBE.Heat.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Transient Thermal Analysis

In this example, using the support files, you will create a Transient Thermal analysis and execute a sequence of python journal commands that will define and solve the analysis in Mechanical.

This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Mechanical_Transient_Thermal_Example_001.agdb
- Mechanical_Transient_Thermal_Example_001.xml
- Mechanical_Transient_Thermal_Example_001.py

These files are available [here](#).

Procedure

1. Open Workbench and insert a **Transient Thermal** system into the Project Schematic.
2. Double-click the **Engineering Data** cell to open the workspace.
3. Select **File > Import Engineering Data**, navigate to the proper folder location and select `Mechanical_Transient_Thermal_Example_001.xml`.
4. Return to the **Project** tab.
5. Right-click the **Geometry** cell and select **Import Geometry > Browse** and then navigate to the proper folder location and select `Mechanical_Transient_Thermal_Example_001.agdb`.
6. Right-click the **Geometry** cell and select the **Properties** option.
7. Set the **Analysis Type** property to **2D**. This property is contained in the **Advanced Geometry Options** group.
8. Open Mechanical: right-click the **Model** cell and select **Edit**.
9. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.
10. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select `Mechanical_Transient_Thermal_Example_001.py`.
11. Select the **Run Script** option () from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Scenario1 - Set up Unit System: Set up the Unit Systems to Metric (m, kg, N, s, V, A),
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

# Scenario 2 - Set up the Tree Object Items: Define the parts, connections, mesh, Transient Thermal system, the analysis settings, and the solution.
GEOMETRY = Model.Geometry
PART1 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Surface Body 1'][0]
PART2 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Surface Body 2'][0]
MESH = Model.Mesh
TRANSIENT_THERMAL = DataModel.AnalysisByName("Transient Thermal")
ANALYSIS_SETTINGS = TRANSIENT_THERMAL.AnalysisSettings
SOLUTION = TRANSIENT_THERMAL.Solution

# Scenario 3 - Define Named Selections: Define the Named selections that will be used to define the Mesh Controls, and the material properties.
NS_BODY1 = DataModel.GetObjectsByName("Body1")[0]
NS_BODY2 = DataModel.GetObjectsByName("Body2")[0]
NS_EDGES2 = DataModel.GetObjectsByName("Edges2")[0]
NS_EDGE1 = DataModel.GetObjectsByName("Edge1")[0]
NS_EDGE2 = DataModel.GetObjectsByName("Edge2")[0]

# Scenario 4 Assign materials and define 2D behaviour: Assign materials to bodies. Define 2D behaviour as plane stress.
GEOMETRY.Model2DBehavior = Model2DBehavior.Planestress
PART1.Material = 'Mechanical_Transient_Thermal_Example_001'
PART2.Material = 'Mechanical_Transient_Thermal_Example_001'
PART1.Thickness = Quantity('0.1 [m]')
PART2.Thickness = Quantity('0.1 [m]')

# Scenario 5 - Define Mesh Settings: Assign global mesh element size of 0.1 m. Assign mesh methods Quadrilateral Domains.
MESH.ElementSize = Quantity('0.1 [m]')
MESH_METHOD01 = MESH.AddAutomaticMethod()
MESH_METHOD01.Location = NS_BODY1
MESH_METHOD01.Method = MethodType.Automatic
MESH_METHOD02 = MESH.AddAutomaticMethod()
MESH_METHOD02.Location = NS_BODY2
MESH_METHOD02.Method = MethodType.AllTriAllTet

# Scenario 6 - Define Analysis Settings: Set the auto time stepping to off with time step of 7 s. Set step end time to 700 s.
ANALYSIS_SETTINGS.StepEndTime = Quantity('700 [s]')
ANALYSIS_SETTINGS.AutomaticTimeStepping = AutomaticTimeStepping.Off
ANALYSIS_SETTINGS.TimeStep = Quantity('7 [s]')

# Scenario 7 - Insert Thermal loads: Define Initial Temperature. Insert and define Convection and Internal Heat Generation.
INITIAL_TEMPERATURE = TRANSIENT_THERMAL.Children[0]
INITIAL_TEMPERATURE.InitialTemperatureValue = Quantity('50 [C]')

TEMPERATURE = TRANSIENT_THERMAL.AddTemperature()
TEMPERATURE.Location = NS_EDGES2
INITIAL_TEMPERATURE.InitialTemperatureValue = Quantity('50 [C]')

CONVECTION = TRANSIENT_THERMAL.AddConvection()
CONVECTION.Location = NS_EDGE1
CONVECTION.FilmCoefficient.Output.DiscreteValues=[Quantity('10 [W m^-1 m^-1 C^-1]'), Quantity('10 [W m^-1 m^-1 C^-1]')]
CONVECTION.AmbientTemperature.Output.DiscreteValues=[Quantity('20 [C]'), Quantity('20 [C]')]

INTERNAL_HEAT_GENERATION = TRANSIENT_THERMAL.AddInternalHeatGeneration()
INTERNAL_HEAT_GENERATION.Location = NS_BODY2
INTERNAL_HEAT_GENERATION.Magnitude.Inputs[0].DiscreteValues = [Quantity("0 [s]"), Quantity("10 [s]"), Quantity("30 [s]")]
INTERNAL_HEAT_GENERATION.Magnitude.Output.DiscreteValues = [Quantity(10000, "W m^-1 m^-1 m^-1"), Quantity(12000, "W m^-1 m^-1 m^-1")]

HEAT_FLUX = TRANSIENT_THERMAL.AddHeatFlux()
HEAT_FLUX.Location = NS_EDGE2
```

```

HEAT_FLUX.Magnitude.Inputs[0].DiscreteValues = [Quantity("0 [s]"), Quantity("10 [s]"), Quantity("30 [s]"), Quantity("50 [s]")]
HEAT_FLUX.Magnitude.Output.DiscreteValues = [Quantity(5000, "W m^-1 m^-1"), Quantity(6250, "W m^-1 m^-1"), Quantity(7500, "W m^-1 m^-1")]

# Scenario 8 - Insert Results: Insert a Temperature results.

TEMPERATURE01 = SOLUTION.AddTemperature()
TEMPERATURE01.Location = NS_EDGE2
TEMPERATURE01.DisplayTime = Quantity('200[s]')
TEMPERATURE02 = SOLUTION.AddTemperature()
TEMPERATURE02.Location = NS_EDGE2

# Scenario 9 - Solve and Define the Results: Solve the system and set the results variables. Note the Scripting window.

SOLUTION.Solve(True)

TEMPERATURE01_MAX = TEMPERATURE01.Maximum.Value
TEMPERATURE02_MAX = TEMPERATURE02.Maximum.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Coupled Field Static Analysis

In this example, using the support files, you will insert a [Coupled Field Static](#) analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will define and solve the analysis.

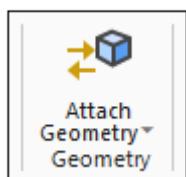
This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Coupled_Field_Static_Example.agdb
 - Coupled_Field_Static_Example.py

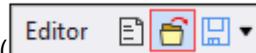
These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
 2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Coupled Field Static** system into the tree.
 3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the [Geometry Context tab](#). Navigate to the proper folder location and select `Coupled_Field_Static_Example.agdb`.



4. Select the Automation tab and select the **Scripting** option to open the **Mechanical Scripting** pane.



5. Select the **Open Script** option () from the [Editor \(p. 4\)](#) toolbar. Navigate to the proper folder location and select `Coupled_Field_Static_Example.py`.



6. Select the **Run Script** option () from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Scenario 1 Set up the Tree Object Items
GEOMETRY = Model.Geometry
COORDINATE_SYSTEM = Model.CoordinateSystems
CONNECTIONS = Model.Connections
MESH = Model.Mesh
COUPLED_FIELD_STATIC = ExtAPI.DataModel.AnalysisByName("Coupled Field Static")
ANALYSIS_SETTINGS = COUPLED_FIELD_STATIC.AnalysisSettings
SOLUTION = COUPLED_FIELD_STATIC.Solution

# Scenario 2 Set up the Unit Systems
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardNMM

# Scenario 3 Define Geometry Details
ROD_PART = GEOMETRY.Children[0].Children[0]
RIGID_PART = GEOMETRY.Children[1].Children[0]
RIGID_PART.StiffnessBehavior=StiffnessBehavior.Rigid

# Scenario 4 Define Named Selections
NS_FIXED_ROD = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Fixed_Rod'][0]
NS_FIXED_RIGID = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Fixed_Rigid'][0]
NS_RIGID_EDGES1 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Rigid_Edges1'][0]
NS_RIGID_EDGES2 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Rigid_Edges2'][0]
NS_RIGID_FACE = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Rigid_Face'][0]
NS_ROD_EDGES1 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Rod_Edges1'][0]
NS_ROD_EDGES2 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Rod_Edges2'][0]
NS_ROD_FACE = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Rod_Face'][0]
NS_ROD_BODY = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Rod_Body'][0]
NS_RIGID_BODY = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Rigid_Body'][0]
NS_BOTH_BODIES = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Both_Bodies'][0]

# Scenario 5 Insert Local Coordinate System
LCS = COORDINATE_SYSTEM.AddCoordinateSystem()
LCS.OriginLocation = NS_ROD_BODY
LCS.CreateConstructionSurface()
CONST_GEOM = ExtAPI.DataModel.Project.Model.ConstructionGeometry
CONST_SURF = CONST_GEOM.Children[0]

# Scenario 6 Define Contact
CONTACTS = CONNECTIONS.Children[0]
CONTACTS_REGION = CONTACTS.Children[0]
CONTACTS_REGION.ContactType=ContactType.Frictional
CONTACTS_REGION.FrictionCoefficient=0.3
CONTACTS_REGION.TimeStepControls=ContactTimeStepControls.PredictForImpact
CONTACTS_REGION.NormalStiffnessValueType=ElementControlsNormalStiffnessType.AbsoluteValue
CONTACTS_REGION.NormalStiffnessValue=Quantity('1e10[N mm^-1 mm^-1]')

# Scenario 7 Define Mesh Settings
MESH_ElementSize = Quantity('25 [mm]')
MESH_ElementOrder = ElementOrder.Quadratic
MESH_METHOD = MESH.AddAutomaticMethod()
MESH_METHOD.Location = NS_ROD_BODY
MESH_METHOD.Method = MethodType.HexDominant
MESH.GenerateMesh()
```

```

# Scenario 8 Define Physics Region
PHY_REG01 = COUPLED_FIELD_STATIC.Children[2]
PHY_REG01.Location = NS_ROD_BODY
PHY_REG01.ThermalStrain = ThermalStrainType.Strong
PHY_REG02 = COUPLED_FIELD_STATIC.AddPhysicsRegion()
PHY_REG02.Structural = True

# Scenario 9 Define Analysis Settings
INITIAL_PHYSICS_OPTION = COUPLED_FIELD_STATIC.Children[0]
ANALYSIS_SETTINGS.NumberOfSteps = 1
ANALYSIS_SETTINGS.StepEndTime = Quantity('1 [sec]')
ANALYSIS_SETTINGS.AutomaticTimeStepping = AutomaticTimeStepping.Off
ANALYSIS_SETTINGS.NumberOfSubSteps = 10

# Scenario 10 Insert Load and Boundary Conditions
FIXED_SUPPORT = COUPLED_FIELD_STATIC.AddFixedSupport()
FIXED_SUPPORT.Location = NS_FIXED_ROD

REMOTE_DISPLACEMENT = COUPLED_FIELD_STATIC.AddRemoteDisplacement()
REMOTE_DISPLACEMENT.Location = NS_FIXED_RIGID
REMOTE_DISPLACEMENT.XComponent.Output.DiscreteValues = [Quantity("0 [mm]")]
REMOTE_DISPLACEMENT.YComponent.Output.DiscreteValues = [Quantity("0 [mm]")]
REMOTE_DISPLACEMENT.ZComponent.Output.DiscreteValues = [Quantity("0 [mm]")]
REMOTE_DISPLACEMENT.RotationX.Output.DiscreteValues = [Quantity("0 [deg]")]
REMOTE_DISPLACEMENT.RotationY.Output.DiscreteValues = [Quantity("0 [deg]")]
REMOTE_DISPLACEMENT.RotationZ.Output.DiscreteValues = [Quantity("0 [deg")]

TEMPERATURE = COUPLED_FIELD_STATIC.AddTemperature()
TEMPERATURE.Location = NS_ROD_BODY
TEMPERATURE.Magnitude.Output.DiscreteValues = [Quantity('122 [C]')]

# Scenario 11 Insert Results
DIRECTIONAL_DEFORMATION = SOLUTION.AddDirectionalDeformation()
DIRECTIONAL_DEFORMATION.Location = NS_ROD_BODY
DIRECTIONAL_DEFORMATION.NormalOrientation = NormalOrientationType.ZAxis

NORMAL_STRESS = SOLUTION.AddNormalStress()
NORMAL_STRESS.Location = CONST_SURF
NORMAL_STRESS.NormalOrientation = NormalOrientationType.ZAxis

THERMAL_STRAIN = SOLUTION.AddThermalStrain()
THERMAL_STRAIN.NormalOrientation=NormalOrientationType.ZAxis
THERMAL_STRAIN.Location = NS_ROD_BODY

# Scenario 12 Solve and review the Result
SOLUTION.Solve(True)

MAXIMUM_DIRECTIONAL_DEFORIFICATION = DIRECTIONAL_DEFORMATION.Maximum.Value
MINIMUM_DIRECTIONAL_DEFORIFICATION = DIRECTIONAL_DEFORMATION.Minimum.Value
MAXIMUM_NORMAL_STRESS = NORMAL_STRESS.Maximum.Value
MAXIMUM_THERMAL_STRAIN = THERMAL_STRAIN.Maximum.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Coupled Field Harmonic Analysis

In this example, using the support files, you will insert a [Coupled Field Harmonic](#) analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will define and solve the analysis.

This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Coupled_Field_Harmonic_Example.agdb
- Coupled_Field_Harmonic_Example.py
- Coupled_Field_Harmonic_Example_PZT.xml
- Coupled_Field_Harmonic_Example_Substrate.xml

These files are available [here](#).

Procedure

1. Open Workbench and insert a **Coupled Field Harmonic** system into the **Project Schematic**.
2. Double-click the **Engineering Data** cell to open the workspace.
3. Select **File > Import Engineering Data**, navigate to the proper folder location and select Coupled_Field_Harmonic_Example_PZT.xml. Repeat this for Coupled_Field_Harmonic_Example_Substrate.xml.
4. Return to the **Project** tab.
5. Right-click the **Geometry** cell and select **Import Geometry > Browse** and then navigate to the proper folder location and select Coupled_Field_Harmonic_Example.agdb.
6. Open Mechanical: right-click the **Model** cell and select **Edit**.
7. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.

8. Select the **Open Script** option () from the [Editor \(p. 4\)](#) toolbar. Navigate to the proper folder location and select Coupled_Field_Harmonic_Example.py.

9. Select the **Run Script** option () from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Scenario 1 Set up the Tree Object Items
GEOMETRY = Model.Geometry
COORDINATE_SYSTEM = Model.CoordinateSystems
CONNECTIONS = Model.Connections
MESH = Model.Mesh
COUPLED_FIELD_HARMONIC = ExtAPI.DataModel.AnalysisByName("Coupled Field Harmonic")
ANALYSIS_SETTINGS = COUPLED_FIELD_HARMONIC.AnalysisSettings
SOLUTION = COUPLED_FIELD_HARMONIC.Solution

# Scenario 2 Set up the Unit Systems
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

# Scenario 3 Assign Material Properties
SUBSTRATE = GEOMETRY.Children[0].Children[0]
```

```

PIEZO = GEOMETRY.Children[1].Children[0]
SUBSTRATE.Assignment='substrate_mat'
PIEZO.Assignment='PZT-5A'

# Scenario 4 Define Named Selections
PIEZO_PART = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'piezo_body'][0]
STRUCT_PART = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'struct_body'][0]
VERTEX = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'tip'][0]
DISP_BC = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'disp_constraints'][0]
PIEZO_TOP = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'piezo_top_surface'][0]
PIEZO_BOTTOM= [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'piezo_bottom_surface'][0]

# Scenario 5 Define Contact
CONTACTS = CONNECTIONS.Children[0]
CONTACTS_REGION = CONTACTS.Children[0]
CONTACTS_REGION.ContactFormulation=ContactFormulation.MPC

# Scenario 6 Define Mesh Settings
MESH.ElementSize = Quantity(' 0.005 [m]')
MESH.GenerateMesh()

# Scenario 7 Define Physics Region
PHY_REG01 = COUPLED_FIELD_HARMONIC.Children[2]
PHY_REG01.Location = PIEZO_PART
PHY_REG02 = COUPLED_FIELD_HARMONIC.AddPhysicsRegion()
PHY_REG02.Structural = True
PHY_REG02.Electric=ElectricType.No

# Scenario 8 Define Analysis Settings
ANALYSIS_SETTINGS.RangeMaximum = Quantity('125[Hz]')
ANALYSIS_SETTINGS.SolutionIntervals = 25

# Scenario 9 Insert Load and Boundary Conditions
DISPLACEMENT = COUPLED_FIELD_HARMONIC.AddDisplacement()
DISPLACEMENT.Location = DISP_BC
DISPLACEMENT.XComponent.Output.DiscreteValues=[Quantity('0 [m]')]
DISPLACEMENT.YComponent.Output.DiscreteValues=[Quantity('0 [m]')]
DISPLACEMENT.ZComponent.Output.DiscreteValues=[Quantity('0 [m]')]

VOLTAGE_COUPLING = COUPLED_FIELD_HARMONIC.AddVoltageCoupling()
VOLTAGE_COUPLING.Location = PIEZO_TOP

VOLTAGE_GROUND = COUPLED_FIELD_HARMONIC.AddVoltageGround()
VOLTAGE_GROUND.Location = PIEZO_BOTTOM

VOLTAGE = COUPLED_FIELD_HARMONIC.AddVoltage()
VOLTAGE.Location = VOLTAGE_COUPLING
VOLTAGE.Magnitude.Output.DiscreteValues = [Quantity('1 [V]')]

# Scenario 10 Insert Results
FRQUENCY_RESPONSE_DEFORMATION = SOLUTION.AddDeformationFrequencyResponse()
FRQUENCY_RESPONSE_DEFORMATION.Location = VERTEX
FRQUENCY_RESPONSE_DEFORMATION.NormalOrientation = NormalOrientationType.ZAxis

FREQUENCY_RESPONSE_VOLTAGE = SOLUTION.AddVoltageFrequencyResponse()
FREQUENCY_RESPONSE_VOLTAGE.Location = PIEZO_TOP

FREQUENCY_RESPONSE_CHARGEREACTION = SOLUTION.AddChargeReactionFrequencyResponse()
FREQUENCY_RESPONSE_CHARGEREACTION.BoundaryCondition = VOLTAGE

FREQUENCY_RESPONSE_IMPEDANCE = SOLUTION.AddImpedanceFrequencyResponse()
FREQUENCY_RESPONSE_IMPEDANCE.BoundaryCondition = VOLTAGE

# Scenario 11 Solve and review the Result
SOLUTION.Solve(True)
FRQUENCY_RESPONSE_DEFORMATION.MaximumAmplitude.Value
FRQUENCY_RESPONSE_DEFORMATION.FrequencyAtMaximumAmplitude.Value

FREQUENCY_RESPONSE_VOLTAGE.MaximumAmplitude.Value
FREQUENCY_RESPONSE_VOLTAGE.FrequencyAtMaximumAmplitude.Value
FREQUENCY_RESPONSE_VOLTAGE.PhaseAngle.Value

```

```

FREQUENCY_RESPONSE_VOLTAGE.RealAtMaximumAmplitude.Value
FREQUENCY_RESPONSE_VOLTAGE.ImaginaryAtMaximumAmplitude.Value

FREQUENCY_RESPONSE_CHARGE_REACTION.MaximumAmplitude.Value
FREQUENCY_RESPONSE_CHARGE_REACTION.FrequencyAtMaximumAmplitude.Value
FREQUENCY_RESPONSE_CHARGE_REACTION.PhaseAngle.Value
FREQUENCY_RESPONSE_CHARGE_REACTION.RealAtMaximumAmplitude.Value
FREQUENCY_RESPONSE_CHARGE_REACTION.ImaginaryAtMaximumAmplitude.Value

FREQUENCY_RESPONSE_IMPEDANCE.MaximumAmplitude.Value
FREQUENCY_RESPONSE_IMPEDANCE.FrequencyAtMaximumAmplitude.Value
FREQUENCY_RESPONSE_IMPEDANCE.PhaseAngle.Value
FREQUENCY_RESPONSE_IMPEDANCE.RealAtMaximumAmplitude.Value
FREQUENCY_RESPONSE_IMPEDANCE.ImaginaryAtMaximumAmplitude.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Coupled Field Transient Analysis

In this example, using the support files, you will insert a **Coupled Field Transient** analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will define and solve the analysis.

This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Coupled_Field_Transient_Example.agdb
- Coupled_Field_Transient_Example.py
- Coupled_Field_Transient_Example_Mat.xml

These files are available [here](#).

Procedure

1. Open Workbench and insert a **Coupled Field Transient** system into the **Project Schematic**.
2. Double-click the **Engineering Data** cell to open the workspace.
3. Select **File > Import Engineering Data**, navigate to the proper folder location and select `Coupled_Field_Transient_Example_Mat.xml`.
4. Right-click the **Geometry** cell and select **Properties**.
5. Set the **Analysis Type** property to **2D** under the **Advanced Geometry Options** category.
6. Right-click the **Geometry** cell and select **Import Geometry > Browse** and then navigate to the proper folder location and select `Coupled_Field_Transient_Example.agdb`.
7. Open Mechanical: right-click the **Model** cell and select **Edit**.
8. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.

9. Select the **Open Script** option ( from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select `Coupled_Field_Transient_Example.py`.

10. Select the **Run Script** option ( from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Scenario 1 Set up the Tree Object Items
GEOMETRY = Model.Geometry
COORDINATE_SYSTEM = Model.CoordinateSystems
CONNECTIONS = Model.Connections
MESH = Model.Mesh
COUPLED_FIELD_TRANSIENT = ExtAPI.DataModel.AnalysisByName("Coupled Field Transient")
ANALYSIS_SETTINGS = COUPLED_FIELD_TRANSIENT.AnalysisSettings
SOLUTION = COUPLED_FIELD_TRANSIENT.Solution

# Scenario 2 Set up the Unit Systems
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS
ExtAPI.Application.ActiveMetricTemperatureUnit = MetricTemperatureUnitType.Kelvin

# Scenario 3 Define Geometry Details
GEOMETRY.Model2DBehavior=Model2DBehavior.AxiSymmetric

# Scenario 4 Define Named Selections
BODY_NS = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'body'][0]
VERT_EDGE1_NS = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'vert_edge1'][0]
VERT_EDGE2_NS = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'vert_edge2'][0]
HORZ_EDGE1_NS = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'horz_edge1'][0]
HORZ_EDGE2_NS = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'horz_edge2'][0]
SURFACE_NS = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'surface'][0]

# Scenario 5 Assign Material Properties
SURFACE1 = GEOMETRY.Children[0].Children[0]
SURFACE1.Assignment ='MAT1'

# Scenario 6 Define Mesh Settings
MESH.ElementOrder=ElementOrder.Linear
MESH_SIZE1 = MESH.AddSizing()
MESH_SIZE1.Location=HORZ_EDGE1_NS
MESH_SIZE1.Type=SizingType.NumberOfDivisions
MESH_SIZE1.NumberOfDivisions=10
MESH_SIZE1.Behavior=SizingBehavior.Hard
MESH_SIZE2 = MESH_SIZE1.Duplicate()
MESH_SIZE2.Location=HORZ_EDGE2_NS
MESH_SIZE3 = MESH.AddSizing()
MESH_SIZE3.Location=VERT_EDGE1_NS
MESH_SIZE3.Type=SizingType.NumberOfDivisions
MESH_SIZE3.NumberOfDivisions=1
MESH_SIZE3.Behavior=SizingBehavior.Hard
MESH_SIZE4 = MESH_SIZE3.Duplicate()
MESH_SIZE4.Location=VERT_EDGE2_NS
FACE_MESH = MESH.AddFaceMeshing()
FACE_MESH.Location=SURFACE_NS
MESH.GenerateMesh()

# Scenario 7 Define Initial Physics Option
INITIAL_PHYSICS_OPTION = COUPLED_FIELD_TRANSIENT.Children[1]
INITIAL_PHYSICS_OPTION.InitialTemperatureValue=Quantity('293[K]')
INITIAL_PHYSICS_OPTION.ReferenceTemperature=Quantity('293[K]')

# Scenario 8 Define Analysis Settings
ANALYSIS_SETTINGS.DefineBy=TimeStepDefineByType.Substeps
ANALYSIS_SETTINGS.InitialSubsteps=20
```

```

ANALYSIS_SETTINGS.MinimumSubsteps=5
ANALYSIS_SETTINGS.MaximumSubsteps=100
ANALYSIS_SETTINGS.StepEndTime=Quantity('130[s]')
ANALYSIS_SETTINGS.HeatConvergence=ConvergenceToleranceType.On
ANALYSIS_SETTINGS.HeatConvergenceValue=Quantity('0.01[W]')
ANALYSIS_SETTINGS.HeatConvergenceTolerance=1
ANALYSIS_SETTINGS.StructuralOnly=False

# Scenario 9 Insert Load and Boundary Conditions
DISPLACEMENT1 = COUPLED_FIELD_TRANSIENT.AddDisplacement()
DISPLACEMENT1.Location = HORZ_EDGE1_NS.Location
DISPLACEMENT1.YComponent.Output.DiscreteValues = [Quantity('0 [m]')]

DISPLACEMENT2 = COUPLED_FIELD_TRANSIENT.AddDisplacement()
DISPLACEMENT2.Location = HORZ_EDGE2_NS.Location
DISPLACEMENT2.YComponent.Output.DiscreteValues = [Quantity('0 [m]')]

DISPLACEMENT3 = COUPLED_FIELD_TRANSIENT.AddDisplacement()
DISPLACEMENT3.Location = VERT_EDGE1_NS.Location
DISPLACEMENT3.XComponent.Output.DiscreteValues = [Quantity('0.13 [m]')]
DISPLACEMENT3.YComponent.Output.DiscreteValues = [Quantity('0 [m]')]

PLASTIC_HEATING = COUPLED_FIELD_TRANSIENT.AddPlasticHeating()
PLASTIC_HEATING.Location = BODY_NS
PLASTIC_HEATING.PlasticWorkFraction = 0.9

COMMAND_SNIPPET = COUPLED_FIELD_TRANSIENT.AddCommandSnippet()
Cmds_1= "AUTOTS,OFF\nKBC,0\nTINTP,,,1.0\nOUTRES,ALL,ALL"
COMMAND_SNIPPET.AppendText(Cmds_1)

# Scenario 10 Insert Results
TEMPERATURE = SOLUTION.AddTemperature()
EQUIVALENT_PLASTIC_STRAIN = SOLUTION.AddEquivalentPlasticStrain()

# Scenario 11 Solve and review the Result
SOLUTION.Solve(True)

MAXIMUM_TEMPERATURE = TEMPERATURE.Maximum.Value
MINIMUM_TEMPERATURE = TEMPERATURE.Minimum.Value

MAXIMUM_EQUIVALENT_PLASTIC_STRAIN = EQUIVALENT_PLASTIC_STRAIN.Maximum.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Coupled Field Modal Analysis

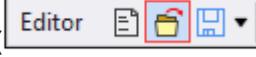
In this example, using the support files, you will insert a **Coupled Field Modal** analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will define and solve the analysis.

This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- `Coupled_Field_Modal_Example.agdb`
- `Coupled_Field_Modal_Example.py`
- `Coupled_Field_Modal_Example_Mat1.xml`

These files are available [here](#).

Procedure

1. Open Workbench and insert a **Coupled Field Modal** system into the **Project Schematic**.
2. Double-click the **Engineering Data** cell to open the workspace.
3. Select **File > Import Engineering Data**, navigate to the proper folder location and select **Coupled_Field_Modal_Example_Mat1.xml**.
4. Return to the **Project** tab.
5. Right-click the **Geometry** cell and select **Import Geometry > Browse** and then navigate to the proper folder location and select **Coupled_Field_Modal_Example.agdb**.
6. Open Mechanical: right-click the **Model** cell and select **Edit**.
7. Select the **Automation** tab and select the **Scripting** option to open the **Mechanical Scripting** pane.
8. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select **Coupled_Field_Modal_Example.py**.
9. Select the **Run Script** option () from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Scenario 1 Set up the Tree Object Items
GEOMETRY = Model.Geometry
COORDINATE_SYSTEM = Model.CoordinateSystems
CONNECTIONS = Model.Connections
MESH = Model.Mesh
COUPLED_FIELD_MODAL = ExtAPI.DataModel.AnalysisByName("Coupled Field Modal")
ANALYSIS_SETTINGS = COUPLED_FIELD_MODAL.AnalysisSettings
SOLUTION = COUPLED_FIELD_MODAL.Solution

# Scenario 2 Set up the Unit Systems
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

# Scenario 3 Assign Material Properties
CUBE = GEOMETRY.Children[0].Children[0]
CUBE.Assignment = 'PZT4'

# Scenario 4 Define Named Selections
SYMM_Y = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'symm_y'][0]
SYMM_X = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'symm_x'][0]
VER_EDGES_N = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'ver_edges'][0]
HOR_EDGES_N = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'hor_edges'][0]
TOP_FACE_N = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'top'][0]
BOT_FACE_N = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'bottom'][0]

# Scenario 5 Insert Symmetry
SYMMETRY = Model.AddSymmetry()
SYMMETRY_REGION01 = SYMMETRY.AddSymmetryRegion()
SYMMETRY_REGION01.Location = SYMM_Y
SYMMETRY_REGION01.SymmetryNormal = SymmetryNormalType.YAxis
SYMMETRY_REGION02 = SYMMETRY.AddSymmetryRegion()
SYMMETRY_REGION02.Location = SYMM_X
SYMMETRY_REGION02.SymmetryNormal = SymmetryNormalType.XAxis
```

```

# Scenario 6 Define Mesh Settings
EDGE_SIZING1 = MESH.AddSizing()
EDGE_SIZING1.Location = VER_EDGES_N
EDGE_SIZING1.Type = SizingType.NumberOfDivisions
EDGE_SIZING1.NumberOfDivisions = 4
EDGE_SIZING2 = MESH.AddSizing()
EDGE_SIZING2.Location = HOR_EDGES_N
EDGE_SIZING2.Type = SizingType.NumberOfDivisions
EDGE_SIZING2.NumberOfDivisions = 2
MESH.GenerateMesh()

# Scenario 7 Define Analysis Settings
ANALYSIS_SETTINGS.MaximumModesToFind = 10
ANALYSIS_SETTINGS.LimitSearchToRange = True
ANALYSIS_SETTINGS.SearchRangeMinimum = Quantity('50000[Hz]')
ANALYSIS_SETTINGS.SearchRangeMaximum = Quantity('150000[Hz]')
ANALYSIS_SETTINGS.SolverType = SolverType.Direct

# Scenario 8 Insert Load and Boundary Conditions
VOLTAGE_GROUND1 = COUPLED_FIELD_MODAL.AddVoltageGround()
VOLTAGE_GROUND1.Location = TOP_FACE_N
VOLTAGE_GROUND2 = COUPLED_FIELD_MODAL.AddVoltageGround()
VOLTAGE_GROUND2.Location = BOT_FACE_N

# Scenario 9 Insert Results
TOTAL_DEFORMATION1 = SOLUTION.AddTotalDeformation()
TOTAL_DEFORMATION1.Mode = 2
TOTAL_DEFORMATION2 = SOLUTION.AddTotalDeformation()
TOTAL_DEFORMATION2.Mode = 5

# Scenario 10 Solve and review the Result
SOLUTION.Solve(True)

TOTAL_DEFORMATION1.ReportedFrequency.Value
TOTAL_DEFORMATION2.ReportedFrequency.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Fracture Analysis: Semi-Elliptical Crack

This example creates a Fracture Analysis that uses a Semi-Elliptical Crack. A semi-elliptical crack is inserted at the tubular joint of the structure, the crack mesh is generated for defined Semi-Elliptical Crack, and then fracture parameters are computed and post-processed.

This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

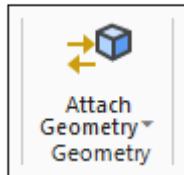
- Semi-Elliptical_Crack_Example_003.agdb
- Semi-Elliptical_Crack_Example_003.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).

2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Static Structural** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the **Geometry Context tab**. Navigate to the proper folder location and select Semi-Elliptical_Crack_Example_003.agdb.



4. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.



5. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select Semi-Elliptical_Crack_Example_003.py.



6. Select the **Run Script** option () from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Scenario 1 Store main Tree Object items
MODEL = Model
GEOMETRY = Model.Geometry
SOLID2 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Solid2'][0]

COORDINATE_SYSTEMS = Model.CoordinateSystems
GLOBAL_COORDINATE_SYSTEM = [i for i in COORDINATE_SYSTEMS.GetChildren[Ansys.ACT.Automation.Mechanical.CoordinateSystem]()]
MESH = Model.Mesh

NAMED_SELECTIONS = ExtAPI.DataModel.Project.Model.NamedSelections
NS_PIPE_BODY = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_PIPE_BODY']
NS_FILET_FACE = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_FILET_FACE']
NS_VERTEX_FILET = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_VERTEX_FILET']
NS_PRESSURE_FACE = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_PRESSURE_FACE']

STATIC_STRUCTURAL = ExtAPI.DataModel.AnalysisByName("Static Structural")
ANALYSIS_SETTINGS = STATIC_STRUCTURAL.AnalysisSettings
SOLUTION= STATIC_STRUCTURAL.Solution

# Scenario 2 Set Display Unit
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardNMM

# Scenario 3 Add local Coordinate System for defining Semi-Elliptical Crack
COORDINATE_SYSTEMS.Activate()
COORDINATE_SYSTEM = COORDINATE_SYSTEMS.AddCoordinateSystem()
SEL=ExtAPI.SelectionManager.AddSelection(NS_VERTEX_FILET)
SEL2=ExtAPI.SelectionManager.CurrentSelection
COORDINATE_SYSTEM.OriginLocation = SEL2
ExtAPI.SelectionManager.ClearSelection()

COORDINATE_SYSTEM.AddTransformation(TransformationType.Rotation, CoordinateSystemAxisType.PositiveZAxis)
COORDINATE_SYSTEM.SetTransformationValue(1, 22.2922)
COORDINATE_SYSTEM.AddTransformation(TransformationType.Rotation, CoordinateSystemAxisType.PositiveYAxis)
COORDINATE_SYSTEM.SetTransformationValue(2, 180)
```



```

TOTAL_DEFORMATION.Activate()
MAX_TOTAL_DEFORMATION = TOTAL_DEFORMATION.Maximum.Value

SIFS_K1.Activate()
MIN_SIFS_K1_C6 = SIFS_K1.Minimum.Value
MAX_SIFS_K1_C6 = SIFS_K1.Maximum.Value

SIFS_K2.Activate()
MIN_SIFS_K2_C6 = SIFS_K2.Minimum.Value
MAX_SIFS_K2_C6 = SIFS_K2.Maximum.Value

SIFS_K3.Activate()
MIN_SIFS_K3_C6 = SIFS_K3.Minimum.Value
MAX_SIFS_K3_C6 = SIFS_K3.Maximum.Value

J_INTEGRAL.Activate()
MIN_J_INTEGRAL_C6 = J_INTEGRAL.Minimum.Value
MAX_J_INTEGRAL_C6 = J_INTEGRAL.Maximum.Value

MATERIAL_FORCE_X.Activate()
MIN_MATERIAL_FORCE_X_C6 = MATERIAL_FORCE_X.Minimum.Value
MAX_MATERIAL_FORCE_X_C6 = MATERIAL_FORCE_X.Maximum.Value

MATERIAL_FORCE_Y.Activate()
MIN_MATERIAL_FORCE_Y_C6 = MATERIAL_FORCE_Y.Minimum.Value
MAX_MATERIAL_FORCE_Y_C6 = MATERIAL_FORCE_Y.Maximum.Value

MATERIAL_FORCE_Z.Activate()
MIN_MATERIAL_FORCE_Z_C6 = MATERIAL_FORCE_Z.Minimum.Value
MAX_MATERIAL_FORCE_Z_C6 = MATERIAL_FORCE_Z.Maximum.Value

T_STRESS.Activate()
MIN_T_STRESS_C6 = T_STRESS.Minimum.Value
MAX_T_STRESS_C6 = T_STRESS.Maximum.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Fracture Analysis: SMART Crack Growth

In this example, using the support files, you will insert a Static Structural analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will define and solve the analysis.

This example begins in the Mechanical application. It requires you to download the following files.

- SMART Crack Growth_Example_033.cdb
- SMART Crack Growth_Example_033.py
- SMART Crack Growth_Example_033.xml

These files are available [here](#).

Procedure

1. Open Workbench and insert a **External Model** system from the **Component System** category of the **Toolbox** and place it in the Project Schematic.

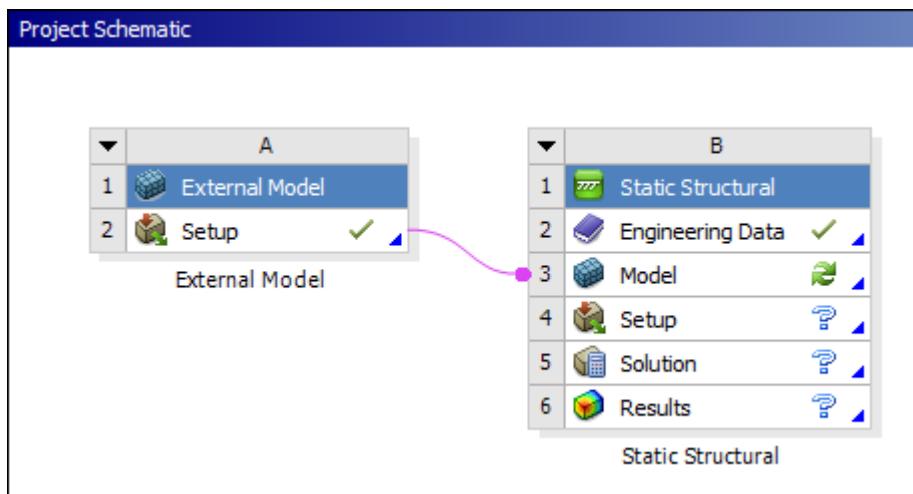
2. Double-click the **Setup** cell to open the workspace.
3. Select the option in the **Location** column to open a dialog. Navigate to the proper folder location and select SMART Crack Growth_Example_033.cdb.

Outline of Schematic A2 : External Model				
	A	B	C	D
1	Data Source ▾	Location	Identifier ▾	Description ▾
2	Click here to add a file	...		

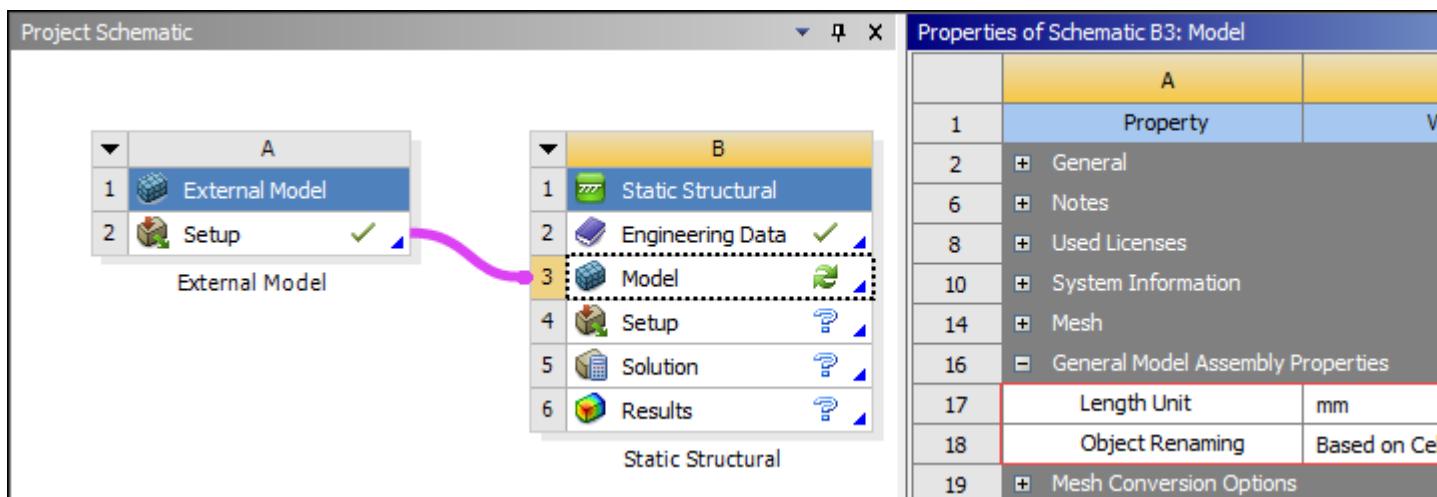
4. Set the **Unit System** property to **Consistent NMM**.

Properties of File - C:\		
	A	B
4	Definition	
5	Unit System	Consistent NMM ▾
6	Process Nodal Components	<input checked="" type="checkbox"/>
7	Nodal Component Key	
8	Process Element Components	<input checked="" type="checkbox"/>
9	Element Component Key	
10	Process Face Components	<input checked="" type="checkbox"/>
11	Face Component Key	
12	Process Model Data	<input checked="" type="checkbox"/>
13	Process Mesh200 Elements	<input type="checkbox"/>
14	Check Valid Blocked CDB File	<input checked="" type="checkbox"/>
15	Node And Element Renumbering Method	Automatic ▾
16	Transformation Type	Rotation and translation ▾

5. Return to the **Project** tab and insert a **Static Structural** system into the Project Schematic.
6. Double-click the **Engineering Data** cell to open the workspace.
7. Select **File > Import Engineering Data**, navigate to the proper folder location and select SMART Crack Growth_Example_033.xml.
8. Return to the **Project** tab. Link the **Setup** cell of the **External Model** system to the **Model** cell of the static analysis.



9. Select the **Model** cell of the static system and display the properties. Set the following properties:
 - Unit Length = mm
 - Object Renaming = Based on Cell ID



10. Right-click the **Model** cell and first select **Update Upstream Components** option and then select **Refresh** option.
11. Open Mechanical. Right-click the **Model** cell and select **Edit**.
12. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.

13. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select SMART_Crack_Growth_Example_033.py.
14. Select the **Run Script** option () from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```

# Scenario 1 Store main Tree Object items
MODEL = Model
GEOMETRY = Model.Geometry
BODY = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Solid Body 1(A2)'][0]

MAT_GRP = MODEL.Materials
MAT_BODY = [i for i in MAT_GRP.GetChildren[Ansys.ACT.Automation.Mechanical.Material](True) if i.Name == 'MAT1'][0]

COORDINATE_SYSTEMS = Model.CoordinateSystems
GLOBAL_COORDINATE_SYSTEM = [i for i in COORDINATE_SYSTEMS.GetChildren[Ansys.ACT.Automation.Mechanical.CoordinateSys]

MESH = Model.Mesh

NAMED_SELECTIONS = ExtAPI.DataModel.Project.Model.NamedSelections
NS_CRACK = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name]
NS_CRACK_SUR01 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name]
NS_CRACK_SUR02 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name]
NS_LOAD01 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name]
NS_LOAD02 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name]
NS_SUPPORT01 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name]
NS_SUPPORT02 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name]

STATIC_STRUCTURAL = ExtAPI.DataModel.AnalysisByName("Static Structural")
ANALYSIS_SETTINGS = STATIC_STRUCTURAL.AnalysisSettings
SOLUTION= STATIC_STRUCTURAL.Solution

# Scenario 2 Set Display Unit
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardNMM

# Scenario 3 Assign Material
BODY.Activate()
BODY.Material = MAT_BODY.Name

# Scenario 4 Rename imported Named Selections
NS_CRACK.Activate()
NS_CRACK.ReadOnly = False
NS_CRACK.Name= "Crack_Front_Nodes"

NS_CRACK_SUR01.Activate()
NS_CRACK_SUR01.ReadOnly = False
NS_CRACK_SUR01.Name= "Crack_TF_Nodes"

NS_CRACK_SUR02.Activate()
NS_CRACK_SUR02.ReadOnly = False
NS_CRACK_SUR02.Name= "Crack_BF_Nodes"

NS_LOAD01.Activate()
NS_LOAD01.ReadOnly = False
NS_LOAD01.Name= "Load_Nodes"

NS_LOAD02.Activate()
NS_LOAD02.ReadOnly = False
NS_LOAD02.Name= "Load2_Nodes"

NS_SUPPORT01.Activate()
NS_SUPPORT01.ReadOnly = False
NS_SUPPORT01.Name= "Support_Nodes"

NS_SUPPORT02.Activate()
NS_SUPPORT02.ReadOnly = False
NS_SUPPORT02.Name= "Support2_Nodes"

# Scenario 5 Add worksheet based Nodal Named Selection
NAMED_SELECTIONSActivate()
NS_SUPPORT03 = NAMED_SELECTIONS.AddNamedSelection()
NS_SUPPORT03.ScopingMethod=GeometryDefineByType.Worksheet

```

```

NS_SUPPORT03.Name = "Support3_Nodes"

GEN_CRT1 = NS_SUPPORT03.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
CRT1.Action=SelectionActionType.Add
CRT1.EntityType=SelectionType.MeshNode
CRT1.Criterion=SelectionCriterionType.LocationZ
CRT1.Operator=SelectionOperatorType.Equal
CRT1.Value=Quantity('0 [mm]')
GEN_CRT1.Add(CRT1)

CRT2 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT2.Active=True
CRT2.Action=SelectionActionType.Add
CRT2.EntityType=SelectionType.MeshNode
CRT2.Criterion=SelectionCriterionType.LocationZ
CRT2.Operator=SelectionOperatorType.Equal
CRT2.Value=Quantity('-12 [mm]')
GEN_CRT1.Add(CRT2)

NS_SUPPORT03.Activate()
NS_SUPPORT03.Generate()

# Scenario 6 Add Pre-Meshed Crack object
MODELActivate()
FRACTURE = MODEL.AddFracture()

FRACTURE.Activate()
PRE_MESHED_CRACK = FRACTURE.AddPreMeshedCrack()
PRE_MESHED_CRACK.CrackFront=NS_CRACK
PRE_MESHED_CRACK.CrackFacesNodes= True
PRE_MESHED_CRACK.TopFaceNodes=NS_CRACK_SUR01
PRE_MESHED_CRACK.BottomFaceNodes=NS_CRACK_SUR02
PRE_MESHED_CRACK.CoordinateSystem = GLOBAL_COORDINATE_SYSTEM
PRE_MESHED_CRACK.SolutionContours = 5
PRE_MESHED_CRACK.CrackSymmetry = False

# Scenario 7 Add SMART Crack Growth object
FRACTURE.Activate()
SMART_CRACK_GROWTH = FRACTURE.AddSMARTCrackGrowth()
SMART_CRACK_GROWTH.InitialCrack = PRE_MESHED_CRACK
SMART_CRACK_GROWTH.CrackGrowthOption = CrackGrowthOption.Fatigue
SMART_CRACK_GROWTH.Material = MAT_BODY.Name
SMART_CRACK_GROWTH.CrackGrowthMethodology = CrackGrowthMethodology.LifeCyclePrediction
SMART_CRACK_GROWTH.MaxIncrementOfCrackExtension = CrackGrowthIncrementType.Manual
SMART_CRACK_GROWTH.MaxIncrementValue = Quantity('0.5 [mm]')
SMART_CRACK_GROWTH.StressRatio = 0

# Scenario 8 Define Analysis Settings
ANALYSIS_SETTINGS.Activate()
ANALYSIS_SETTINGS.AutomaticTimeStepping=AutomaticTimeStepping.Off
ANALYSIS_SETTINGS.DefineBy=TimeStepDefineByType.Substeps
ANALYSIS_SETTINGS.NumberOfSubSteps = 6
ANALYSIS_SETTINGS.JIntegral = False

# Scenario 9 Define boundary conditions
STATIC_STRUCTURAL.Activate()
NODAL_DISPLACEMENT01 = STATIC_STRUCTURAL.AddNodalDisplacement()
NODAL_DISPLACEMENT01.Location = NS_SUPPORT01
NODAL_DISPLACEMENT01.YComponent.Output.DiscreteValues = [Quantity('0 [mm]')]

STATIC_STRUCTURAL.Activate()
NODAL_DISPLACEMENT02 = STATIC_STRUCTURAL.AddNodalDisplacement()
NODAL_DISPLACEMENT02.Location = NS_SUPPORT02
NODAL_DISPLACEMENT02.XComponent.Output.DiscreteValues = [Quantity('0 [mm]')]

STATIC_STRUCTURAL.Activate()
NODAL_DISPLACEMENT03 = STATIC_STRUCTURAL.AddNodalDisplacement()
NODAL_DISPLACEMENT03.Location = NS_SUPPORT03
NODAL_DISPLACEMENT03.ZComponent.Output.DiscreteValues = [Quantity('0 [mm]')]

```



```
EQV_SIFS_RANGE_PROBE02.Activate()
MIN_VOT_EQV_SIFS_RANGE_P02 = EQV_SIFS_RANGE_PROBE02.MinimumValueOverTime.Value
MAX_VOT_EQV_SIFS_RANGE_P02 = EQV_SIFS_RANGE_PROBE02.MaximumValueOverTime.Value
```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Fracture Analysis: Contact Debonding

In this example, using the support files, you will create a Static Structural analysis, execute a sequence of python journal commands that will setup and solve a Contact Debonding analysis.

This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Contact_Debonding_Example.agdb
- Contact_Debonding_Example.py
- Contact_Debonding_Example_Mat1.xml
- Contact_Debonding_Example_Mat2.xml

These files are available [here](#).

Procedure

1. Open Workbench and insert a **Static Structural** system into the **Project Schematic**.
2. Double-click the **Engineering Data** cell to open the workspace.
3. Select **File > Import Engineering Data**, navigate to the proper folder location and select Contact_Debonding_Example_Mat1.xml. Repeat this for Contact_Debonding_Example_Mat2.xml
4. Return to the Project tab.
5. Right-click the **Geometry** cell and select **Properties**.
6. Enable the **Named Selections** check box under the **Basic Geometry Options** category.
7. Set the **Analysis Type** property to **2D** under the **Advanced Geometry Options** category.
8. Right-click the **Geometry** cell and select **Import Geometry > Browse** and then navigate to the proper folder location and select Contact_Debonding_Example.agdb.
9. Open Mechanical: right-click the **Model** cell and select **Edit**.
10. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.

11. Select the **Open Script** option (from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select `Contact_Debonding_Example.py`.

12. Select the **Run Script** option (from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```

# Scenario 1 Store main Tree Object items
MODEL = Model
GEOMETRY = Model.Geometry
PART = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Part 2'][0]

MAT_GRP = MODEL.Materials
MAT_BODY = [i for i in MAT_GRP.GetChildren[Ansys.ACT.Automation.Mechanical.Material](True) if i.Name == 'Interface'
MAT_CZM = [i for i in MAT_GRP.GetChildren[Ansys.ACT.Automation.Mechanical.Material](True) if i.Name == 'CZM Crack M

CONNECTIONS_GRP = ExtAPI.DataModel.Project.Model.Connections
CONTACTS = [i for i in CONNECTIONS_GRP.GetChildren[Ansys.ACT.Automation.Mechanical.Connections.ConnectionGroup](True)
CONTACT_REGION = [i for i in CONTACTS.GetChildren[Ansys.ACT.Automation.Mechanical.Connections.ContactRegion](True)

MESH = Model.Mesh

NAMED_SELECTIONS = ExtAPI.DataModel.Project.Model.NamedSelections
NS_EDGE_HIGH = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.N
NS_EDGE_LOW = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.N
NS_EDGES_SHORT = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.N
NS_EDGES_LONG = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.N
NS_EDGES_FIXED = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.N
NS_VERTEX_DISP1 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.N
NS_VERTEX_DISP2 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.N
NS_FACES_BOTH = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.N

STATIC_STRUCTURAL = ExtAPI.DataModel.AnalysisByName("Static Structural")
ANALYSIS_SETTINGS = STATIC_STRUCTURAL.AnalysisSettings
SOLUTION= STATIC_STRUCTURAL.Solution

# Scenario 2 Set Display Unit
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardNMM

# Scenario 3 Set 2D Behavior
GEOMETRY.Activate()
GEOMETRY.Model2DBehavior=Model2DBehavior.Planestrain

# Scenario 4 Assign Material
PART.Activate()
PART.Material = MAT_BODY.Name

# Scenario 5 Define Contact Region
CONTACT_REGION.Activate()
CONTACT_REGION.SourceLocation = NS_EDGE_HIGH
CONTACT_REGION.TargetLocation= NS_EDGE_LOW
CONTACT_REGION.ContactType=ContactType.Bonded
CONTACT_REGION.ContactFormulation=ContactFormulation.PurePenalty

# Scenario 6 Define Mesh controls and generate mesh
MESHActivate()
MESH.ElementOrder=ElementOrder.Quadratic
MESH.UseAdaptiveSizing=False
MESH.ElementSize = Quantity('0.75 [mm]')

SIZING_MESH = MESH.AddSizing()
SIZING_MESH.Location = NS_EDGES_SHORT
SIZING_MESH.ElementSize = Quantity('0.75 [mm]')

```

```

SIZING_MESH.Behavior=SizingBehavior.Hard

SIZING_MESH2 = MESH.AddSizing()
SIZING_MESH2.Location = NS_EDGES_LONG
SIZING_MESH2.ElementSize = Quantity('0.5 [mm]')
SIZING_MESH2.Behavior=SizingBehavior.Hard

FACE_MESHING = MESH.AddFaceMeshing()
FACE_MESHING.Location = NS_FACES_BOTH
FACE_MESHING.Method=FaceMeshingMethod.Quadrilaterals

MESH.Activate()
MESH.GenerateMesh()

# Scenario 7 Add Contact Debonding object
MODEL.Activate()
FRACTURE = MODEL.AddFracture()

CONTACT_DEBONDING = FRACTURE.AddContactDebonding()
CONTACT_DEBONDING.Material= MAT_CZM.Name
CONTACT_DEBONDING.ContactRegion=CONTACT_REGION

# Scenario 8 Define Analysis Settings
ANALYSIS_SETTINGS.Activate()
ANALYSIS_SETTINGS.AutomaticTimeStepping=AutomaticTimeStepping.On
ANALYSIS_SETTINGS.DefineBy=TimeStepDefineByType.Substeps
ANALYSIS_SETTINGS.MaximumSubsteps=100
ANALYSIS_SETTINGS.InitialSubsteps=100
ANALYSIS_SETTINGS.MinimumSubsteps=100
ANALYSIS_SETTINGS.LargeDeflection=True

# Scenario 9 Define boundary conditions
STATIC_STRUCTURAL.Activate()
FIXED_SUPPORT = STATIC_STRUCTURAL.AddFixedSupport()
FIXED_SUPPORT.Location = NS_EDGES_FIXED

STATIC_STRUCTURAL.Activate()
DISPLACEMENT = STATIC_STRUCTURAL.AddDisplacement()
DISPLACEMENT.Location = NS_VERTEX_DISP1
DISPLACEMENT.DefineBy = LoadDefineBy.Components
DISPLACEMENT.YComponent.Output.DiscreteValues=[Quantity('10 [mm]')]

STATIC_STRUCTURAL.Activate()
DISPLACEMENT2 = STATIC_STRUCTURAL.AddDisplacement()
DISPLACEMENT2.Location = NS_VERTEX_DISP2
DISPLACEMENT2.DefineBy = LoadDefineBy.Components
DISPLACEMENT2.YComponent.Output.DiscreteValues=[Quantity('-10 [mm]')]

# Scenario 10 Add results
SOLUTION.Activate()
DIRECTIONAL_DEFORMATION = SOLUTION.AddDirectionalDeformation()
DIRECTIONAL_DEFORMATION.NormalOrientation=NormalOrientationType.YAxis

FORCEREACTION = SOLUTION.AddForceReaction()
FORCEREACTION.BoundaryConditionSelection = DISPLACEMENT

# Scenario 11 Solve and review results
STATIC_STRUCTURAL.Activate()
STATIC_STRUCTURAL.Solve(True)

DIRECTIONAL_DEFORMATIONActivate()
MIN_DIRECTIONAL_DEFORMATION = DIRECTIONAL_DEFORMATION.Minimum.Value
MAX_DIRECTIONAL_DEFORMATION = DIRECTIONAL_DEFORMATION.Maximum.Value

FORCEREACTIONActivate()
Y_AXIS_FORCEREACTION = FORCEREACTION.YAxis.Value
MOT_Y_AXIS_FORCEREACTION = FORCEREACTION.MaximumYAxis.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Fracture Analysis: Interface Delamination

In this example, using the support files, you will create a Static Structural analysis, execute a sequence of python journal commands that will setup and solve a Interface Delamination analysis.

This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Interface_Delamination_Example.agdb
- Interface_Delamination_Example.py
- Interface_Delamination_Example_Mat1.xml
- Interface_Delamination_Example_Mat2.xml

These files are available [here](#).

Procedure

1. Open Workbench and insert a **Static Structural** system into the **Project Schematic**.
2. Double-click the **Engineering Data** cell to open the workspace.
3. Select **File > Import Engineering Data**, navigate to the proper folder location and select `Interface_Delamination_Example_Mat1.xml`. Repeat this for `Interface_Delamination_Example_Mat2.xml`
4. Return to the Project tab.
5. Right-click the **Geometry** cell and select **Properties**.
6. Enable the **Named Selections** check box under the **Basic Geometry Options** category.
7. Set the **Analysis Type** property to **2D** under the **Advanced Geometry Options** category.
8. Right-click the **Geometry** cell and select **Import Geometry > Browse** and then navigate to the proper folder location and select `Interface_Delamination_Example.agdb`.
9. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
10. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.
11. Select the **Open Script** option () from the [Editor \(p. 4\)](#) toolbar. Navigate to the proper folder location and select `Interface_Delamination_Example.py`.



12. Select the **Run Script** option () from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```

# Scenario 1 Store main Tree Object items
MODEL = Model
GEOMETRY = Model.Geometry
PART = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Part 2'][0]

MAT_GRP = MODEL.Materials
MAT_BODY = [i for i in MAT_GRP.GetChildren[Ansys.ACT.Automation.Mechanical.Material](True) if i.Name == 'Interface'
MAT_CZM = [i for i in MAT_GRP.GetChildren[Ansys.ACT.Automation.Mechanical.Material](True) if i.Name == 'CZM Material']

COORDINATE_SYSTEMS = Model.CoordinateSystems
GLOBAL_COORDINATE_SYSTEM = [i for i in COORDINATE_SYSTEMS.GetChildren[Ansys.ACT.Automation.Mechanical.CoordinateSystem](True)]

CONNECTIONS_GRP = ExtAPI.DataModel.Project.Model.Connections
CONTACTS = [i for i in CONNECTIONS_GRP.GetChildren[Ansys.ACT.Automation.Mechanical.Connections.ConnectionGroup](True)]
CONTACT_REGION = [i for i in CONTACTS.GetChildren[Ansys.ACT.Automation.Mechanical.Connections.ContactRegion](True)]

MESH = Model.Mesh

NAMED_SELECTIONS = ExtAPI.DataModel.Project.Model.NamedSelections
NS_EDGE_HIGH = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_EDGE_HIGH']
NS_EDGE_LOW = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_EDGE_LOW']
NS_EDGES_SHORT = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_EDGES_SHORT']
NS_EDGES_LONG = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_EDGES_LONG']
NS_EDGES_FIXED = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_EDGES_FIXED']
NS_VERTEX_DISP1 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_VERTEX_DISP1']
NS_VERTEX_DISP2 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_VERTEX_DISP2']
NS_FACES_BOTH = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_FACES_BOTH']

STATIC_STRUCTURAL = ExtAPI.DataModel.AnalysisByName("Static Structural")
ANALYSIS_SETTINGS = STATIC_STRUCTURAL.AnalysisSettings
SOLUTION= STATIC_STRUCTURAL.Solution

# Scenario 2 Set Display Unit
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardNMM

# Scenario 3 Set 2D Behavior
GEOMETRY.Activate()
GEOMETRY.Model2DBehavior=Model2DBehavior.Planestrain

# Scenario 4 Assign Material
PART.Activate()
PART.Material = MAT_BODY.Name

# Scenario 5 Suppress the Contact Region
CONTACT_REGION.Activate()
CONTACT_REGION.Suppressed = True

# Scenario 6 Add local Coordinates
COORDINATE_SYSTEMS.Activate()
COORDINATE_SYSTEM = COORDINATE_SYSTEMS.AddCoordinateSystem()
COORDINATE_SYSTEM.OriginLocation = NS_EDGE_HIGH
COORDINATE_SYSTEM.Name = 'High Coordinate System'

COORDINATE_SYSTEMS.Activate()
COORDINATE_SYSTEM2 = COORDINATE_SYSTEMS.AddCoordinateSystem()
COORDINATE_SYSTEM2.OriginLocation = NS_EDGE_LOW
COORDINATE_SYSTEM2.Name = 'Low Coordinate System'

# Scenario 7 Define Mesh controls and generate mesh
MESH.Activate()
MESH.ElementOrder=ElementOrder.Quadratic

```

```

MESH.UseAdaptiveSizing=False
MESH.ElementSize = Quantity('0.75 [mm]')

MESH.Activate()
MATCH_CONTROL = MESH.AddMatchControl()

SEL=ExtAPI.SelectionManager.AddSelection(NS_EDGE_HIGH)
SEL2=ExtAPI.SelectionManager.CurrentSelection
MATCH_CONTROL.HighGeometrySelection = SEL2
ExtAPI.SelectionManager.ClearSelection()

SEL3=ExtAPI.SelectionManager.AddSelection(NS_EDGE_LOW)
SEL4=ExtAPI.SelectionManager.CurrentSelection
MATCH_CONTROL.LowGeometrySelection = SEL4
ExtAPI.SelectionManager.ClearSelection()

MATCH_CONTROL.Transformation = 1 # For setting to Arbitrary option
MATCH_CONTROL.HighCoordinateSystem = COORDINATE_SYSTEM
MATCH_CONTROL.LowCoordinateSystem = COORDINATE_SYSTEM2
MATCH_CONTROL.Suppressed = True
MATCH_CONTROL.Suppressed = False

SIZING_MESH = MESH.AddSizing()
SIZING_MESH.Location = NS_EDGES_SHORT
SIZING_MESH.ElementSize = Quantity('0.75 [mm]')
SIZING_MESH.Behavior=SizingBehavior.Hard

SIZING_MESH2 = MESH.AddSizing()
SIZING_MESH2.Location = NS_EDGES_LONG
SIZING_MESH2.ElementSize = Quantity('0.5 [mm]')
SIZING_MESH2.Behavior=SizingBehavior.Hard

FACE_MESHING = MESH.AddFaceMeshing()
FACE_MESHING.Location = NS_FACES_BOTH
FACE_MESHING.Method=FaceMeshingMethod.Quadrilaterals

MESH.Activate()
MESH.GenerateMesh()

# Scenario 8 Add Interface Delamination object
MODEL.Activate()
FRACTURE = MODEL.AddFracture()

INTERFACE_DELAMINATION = FRACTURE.AddInterfaceDelamination()
INTERFACE_DELAMINATION.Method=DelaminationMethod.CZM
INTERFACE_DELAMINATION.Material= MAT_CZM.Name
INTERFACE_DELAMINATION.GenerationMethod=DelaminationGenerationMethod.MatchedMeshing
INTERFACE_DELAMINATION.MatchControl=MATCH_CONTROL

# Scenario 9 Define Analysis Settings
ANALYSIS_SETTINGS.Activate()
ANALYSIS_SETTINGS.AutomaticTimeStepping=AutomaticTimeStepping.On
ANALYSIS_SETTINGS.DefineBy=TimeStepDefineByType.Substeps
ANALYSIS_SETTINGS.MaximumSubsteps=40
ANALYSIS_SETTINGS.InitialSubsteps=40
ANALYSIS_SETTINGS.MinimumSubsteps=40
ANALYSIS_SETTINGS.LargeDeflection=True

# Scenario 10 Define boundary conditions
STATIC_STRUCTURALActivate()
FIXED_SUPPORT = STATIC_STRUCTURAL.AddFixedSupport()
FIXED_SUPPORT.Location = NS_EDGES_FIXED

STATIC_STRUCTURAL.Activate()
DISPLACEMENT = STATIC_STRUCTURAL.AddDisplacement()
DISPLACEMENT.Location = NS_VERTEX_DISP1
DISPLACEMENT.DefineBy = LoadDefineBy.Components
DISPLACEMENT.YComponent.Output.DiscreteValues=[Quantity('10 [mm]')]

STATIC_STRUCTURAL.Activate()
DISPLACEMENT2 = STATIC_STRUCTURAL.AddDisplacement()

```

```
DISPLACEMENT2.Location = NS_VERTEX_DISP2
DISPLACEMENT2.DefineBy = LoadDefineBy.Components
DISPLACEMENT2.YComponent.Output.DiscreteValues=[Quantity('-10 [mm]')]

# Scenario 11 Add results
SOLUTION.Activate()
DIRECTIONAL_DEFORMATION = SOLUTION.AddDirectionalDeformation()
DIRECTIONAL_DEFORMATION.NormalOrientation=NormalOrientationType.YAxis

FORCEREACTION = SOLUTION.AddForceReaction()
FORCEREACTION.BoundaryConditionSelection = DISPLACEMENT

# Scenario 12 Solve and review Results
STATIC_STRUCTURAL.Activate()
STATIC_STRUCTURAL.Solve(True)

DIRECTIONAL_DEFORMATION.Activate()
MIN_DIRECTIONAL_DEFORMATION = DIRECTIONAL_DEFORMATION.Minimum.Value
MAX_DIRECTIONAL_DEFORMATION = DIRECTIONAL_DEFORMATION.Maximum.Value

FORCEREACTION.Activate()
Y_AXIS_FORCEREACTION = FORCEREACTION.YAxis.Value
MOT_Y_AXIS_FORCEREACTION = FORCEREACTION.MaximumYAxis.Value
```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Harmonic Acoustic Analysis

In this example, using the attached files, you will insert a **Harmonic Acoustic** analysis system and execute a sequence of python journal commands that will define and solve the analysis.

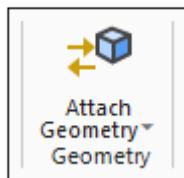
This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Harmonic_Acoustics_Example.agdb
- Harmonic_Acoustics_Example.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Harmonic Acoustics** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the [Geometry Context Tab](#). Navigate to the proper folder location and select **Harmonic_Acoustics_Example.agdb**.



4. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.

5. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select `Harmonic_Acoustics_Example.py`.

6. Select the **Run Script** option () from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
#Scenario 1 Store all main tree nodes as variables
GEOMETRY = Model.Geometry
MESH = Model.Mesh
NAMED_SELECTIONS = Model.NamedSelections
CONNECTIONS = Model.Connections

#Scenario 2 Suppress unneeded bodies and assign material
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

SPEAKER_BOX = [i for i in GEOMETRY.GetChildren[Ansys.ACT.Automation.Mechanical.Body](True) if i.Name == 'speaker-box']
PLATE = [i for i in GEOMETRY.GetChildren[Ansys.ACT.Automation.Mechanical.Body](True) if i.Name == 'plate'][0]
FEA_DOMAIN = [i for i in GEOMETRY.GetChildren[Ansys.ACT.Automation.Mechanical.Body](True) if i.Name == 'FEA_Domain']
PML_REGION = [i for i in GEOMETRY.GetChildren[Ansys.ACT.Automation.Mechanical.Body](True) if i.Name == 'PML_Region']

SPEAKER_BOX.Suppress = 1
PLATE.Suppress = 1

FEA_DOMAIN.Material = "Air"
PML_REGION.Material = "Air"

#Scenario 3 Store Named selections as variable
FEA_BODY = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'FEA_Body']
PML_BODY = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'PML_Body']
MASS_SOURCE_FACE = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'Mass_Source_Face']
PRESSURE_FACES = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'Pressure_Faces']

#Scenario 4 Add mesh sizing and method
SIZING1 = MESH.AddSizing()
SIZING1.Location = FEA_BODY
SIZING1.ElementSize =Quantity('0.01 [m]')

SIZING2 = MESH.AddSizing()
SIZING2.Location = PML_BODY
SIZING2.ElementSize =Quantity('0.01 [m]')

MESH_METHOD1 = MESH.AddAutomaticMethod()
MESH_METHOD1.Location = FEA_BODY.Location

MESH_METHOD1.Method = MethodType.HexDominant

MESH_METHOD2 = MESH.AddAutomaticMethod()
MESH_METHOD2.Location = PML_BODY.Location
```

```

MESH_METHOD2.Method = MethodType.HexDominant

#Scenario 5 Insert automatic node merge
MESH_EDIT = Model.AddMeshEdit()
NODE_MERGE_GROUP = MESH_EDIT.AddNodeMergeGroup()
NODE_MERGE_GROUP.ToleranceValue = Quantity('0.001 [m]')
MESH_EDIT.Generate()

#Scenario 6 Setup Harmonic Acoustics and defined Acoustic and PML regions
ANALYSIS_SETTINGS = Model.Analyses[0].AnalysisSettings
ANALYSIS_SETTINGS.RangeMaximum = Quantity('3000 [Hz]')
ANALYSIS_SETTINGS.SolutionIntervals = 1
ANALYSIS_SETTINGS.CalculateVelocity = True
ANALYSIS_SETTINGS.CalculateEnergy = True
ANALYSIS_SETTINGS.GeneralMiscellaneous = True
ANALYSIS_SETTINGS.FarFieldRadiationSurface = FarFieldRadiationSurfaceType.Manual

HARMONIC_ACOUSTICS = Model.Analyses[0]
ACOUSTIC_REGION1 = HARMONIC_ACOUSTICS.Children[2]
ACOUSTIC_REGION1.Location = FEA_BODY

ACOUSTIC_REGION2 = HARMONIC_ACOUSTICS.AddPhysicsRegion()
ACOUSTIC_REGION2.Location = PML_BODY
ACOUSTIC_REGION2.Acoustics = True
ACOUSTIC_REGION2.ArtificiallyMatchedLayers = ArtificiallyMatchedLayers.PML

MASS_SOURCE = HARMONIC_ACOUSTICS.AddAcousticMassSource()
MASS_SOURCE.Location = MASS_SOURCE_FACE
MASS_SOURCE.Magnitude.Output.DiscreteValues = [Quantity('0.01 [kg m-2 s-1]')]

ACOUST_PRESSURE = HARMONIC_ACOUSTICS.AddAcousticPressure()
ACOUST_PRESSURE.Location = PRESSURE_FACES

FARFIELD_RAD_SURFACE = HARMONIC_ACOUSTICS.CreateAutomaticFarFieldRadiationSurfaces()

#Scenario 7 Insert Acoustic results and solve
SOLUTION = Model.Analyses[0].Solution

ACOUSTIC_PRESSURE_RESULT = SOLUTION.AddAcousticPressureResult()
ACOUSTIC_PRESSURE_RESULT.Location = FEA_BODY

SOUND_PRESSURE_LEVEL = SOLUTION.AddAcousticSoundPressureLevel()
SOUND_PRESSURE_LEVEL.Location = FEA_BODY

FAR_FIELD_SPL = SOLUTION.AddAcousticFarFieldSPL()
FAR_FIELD_SPL.SphereRadius = Quantity('1 [m]')
FAR_FIELD_SPL.ThetaAngleEnd = Quantity('90 [deg]')
FAR_FIELD_SPL.ThetaAngleNoOfDivisions = 90

FAR_FIELD_AWEIGHTED_SPL = SOLUTION.AddAcousticFarFieldAWeightedSPL()
FAR_FIELD_AWEIGHTED_SPL.SphereRadius = Quantity('1 [m]')
FAR_FIELD_AWEIGHTED_SPL.ThetaAngleEnd = Quantity('90 [deg]')
FAR_FIELD_AWEIGHTED_SPL.ThetaAngleNoOfDivisions = 90

#Scenario 8 Solve and store results
SOLUTION.Solve(True)

PRESSURE_MAX = ACOUSTIC_PRESSURE_RESULT.Minimum.Value
PRESSURE_MIN = ACOUSTIC_PRESSURE_RESULT.Maximum.Value

SPL_MIN = SOUND_PRESSURE_LEVEL.Minimum.Value
SPL_MAX = SOUND_PRESSURE_LEVEL.Maximum.Value

FF_SPL_MIN = FAR_FIELD_SPL.Minimum.Value
FF_SPL_MAX = FAR_FIELD_SPL.Maximum.Value

FF_A_SPL_MIN = FAR_FIELD_AWEIGHTED_SPL.Minimum.Value
FF_A_SPL_MAX = FAR_FIELD_AWEIGHTED_SPL.Maximum.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Modal Acoustic Analysis

In this example, using the attached files, you will insert a **Modal Acoustic** analysis system and execute a sequence of python journal commands that will define and solve the analysis.

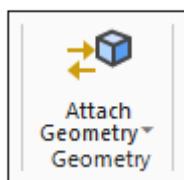
This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Mechanical_Modal_Acoustics_013_Geometry.agdb
- Mechanical_Modal_Acoustics_013_Script.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Modal Acoustics** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the [Geometry Context tab](#). Navigate to the proper folder location and select Mechanical_Modal_Acoustics_013_Geometry.agdb.



4. Select the [Automation tab](#) and select the **Scripting** option to open the **Mechanical Scripting** pane.

5. Select the **Open Script** option (from the [Editor \(p. 4\)](#) toolbar. Navigate to the proper folder location and select Mechanical_Modal_Acoustics_013_Script.py.

6. Select the **Run Script** option (from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
#Scenario 1 Define main Tree node objects
GEOMETRY = Model.Geometry
MESH = Model.Mesh
```

```

NS_GRP = Model.NamedSelections

#Scenario 2 Store Named selections as variables
NS_OUTER_FACE = [i for i in NS_GRP.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_OUTER_FACE']
NS_FSI_FACE = [i for i in NS_GRP.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'FSI_FACE']
NS_ACOUSTIC_BODY = [i for i in NS_GRP.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_ACOUSTIC_BODY']
NS_SOLID_BODY = [i for i in NS_GRP.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_SOLID_BODY']
NS_SOLID_FACE1 = [i for i in NS_GRP.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_SOLID_FACE1']
NS_ACST_FACE1 = [i for i in NS_GRP.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == 'NS_ACST_FACE1']

#Scenario 3 Assign Water to Acoustic parts
GEOM_WATER = [i for i in GEOMETRY.GetChildren[Ansys.ACT.Automation.Mechanical.Body](True) if i.Name == 'Water'][0]

GEOM_WATER.Material = 'Water Liquid'

#Scenario 4 Insert and setup mesh controls
MESH.ElementOrder = ElementOrder.Quadratic

FACE_MESH1 = MESH.AddFaceMeshing()
FACE_MESH1.Location = NS_SOLID_FACE1

FACE_MESH2 = MESH.AddFaceMeshing()
FACE_MESH2.Location = NS_ACST_FACE1

MESH_SIZE1 = MESH.AddSizing()
MESH_SIZE1.Location = NS_SOLID_BODY
MESH_SIZE1.ElementSize = Quantity('0.125 [in]')

MESH_SIZE2 = MESH.AddSizing()
MESH_SIZE2.Location = NS_ACOUSTIC_BODY
MESH_SIZE2.ElementSize = Quantity('5 [in]')

#Scenario 5 Define Physics Regions
MODAL_ACOUSTIC = Model.Analyses[0]

ANALYSIS_SETTINGS = Model.Analyses[0].AnalysisSettings
ANALYSIS_SETTINGS.IgnoreAcousticDamping = True

ACOUSTIC_REGION = MODAL_ACOUSTIC.Children[2]
ACOUSTIC_REGION.Location = NS_ACOUSTIC_BODY
ACOUSTIC_REGION.Acoustics = True

STRUCTURAL_REGION = MODAL_ACOUSTIC.AddPhysicsRegion()
STRUCTURAL_REGION.Location = NS_SOLID_BODY
STRUCTURAL_REGION.Structural = True

#Scenario 6 Insert Acoustic Pressure and FSI
ACOUSTIC_PRESSURE = MODAL_ACOUSTIC.AddAcousticPressure()
ACOUSTIC_PRESSURE.Location = NS_OUTER_FACE
ACOUSTIC_PRESSURE.Magnitude = Quantity('0.000001 [psi]')

FLUID_SOLID_INTERFACE = MODAL_ACOUSTIC.AddFluidSolidInterface()
FLUID_SOLID_INTERFACE.Location = NS_FSI_FACE

#Scenario 7 Insert results Total Deformation and Acoustic Pressure
SOLUTION = MODAL_ACOUSTIC.Solution
TOTAL_DEFORMATION_1 = SOLUTION.AddTotalDeformation()

ACOUSTIC_PRESSURE_RESULT_1 = SOLUTION.AddAcousticPressureResult()
ACOUSTIC_PRESSURE_RESULT_1.Location = NS_ACOUSTIC_BODY

ACOUSTIC_PRESSURE_RESULT_2 = SOLUTION.AddAcousticPressureResult()
ACOUSTIC_PRESSURE_RESULT_2.SetNumber = 2

#Scenario 8 Solve and store results
SOLUTION.Solve(True)

#Frequency for particular mode from details view
FREQ1 = TOTAL_DEFORMATION_1.ReportedFrequency.Value

#Frequency for all modes from tabular data

```

```

FREQ1 = TOTAL_DEFORMATION_1.TabularData["Frequency"][0]
FREQ2 = TOTAL_DEFORMATION_1.TabularData["Frequency"][1]
FREQ3 = TOTAL_DEFORMATION_1.TabularData["Frequency"][2]
FREQ4 = TOTAL_DEFORMATION_1.TabularData["Frequency"][3]
FREQ5 = TOTAL_DEFORMATION_1.TabularData["Frequency"][4]
FREQ6 = TOTAL_DEFORMATION_1.TabularData["Frequency"][5]

PRESSURE_MAX = ACOUSTIC_PRESSURE_RESULT_1.Maximum.Value
PRESSURE_MIN = ACOUSTIC_PRESSURE_RESULT_1.Minimum.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Structural Optimization (Density Based) Analysis

In this example, using the support files, you perform a **Structural Optimization** analysis in Mechanical using a sequence of python journal commands that will define and solve the analysis. This analysis uses the **Topology Optimization - Density Based** method.

This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

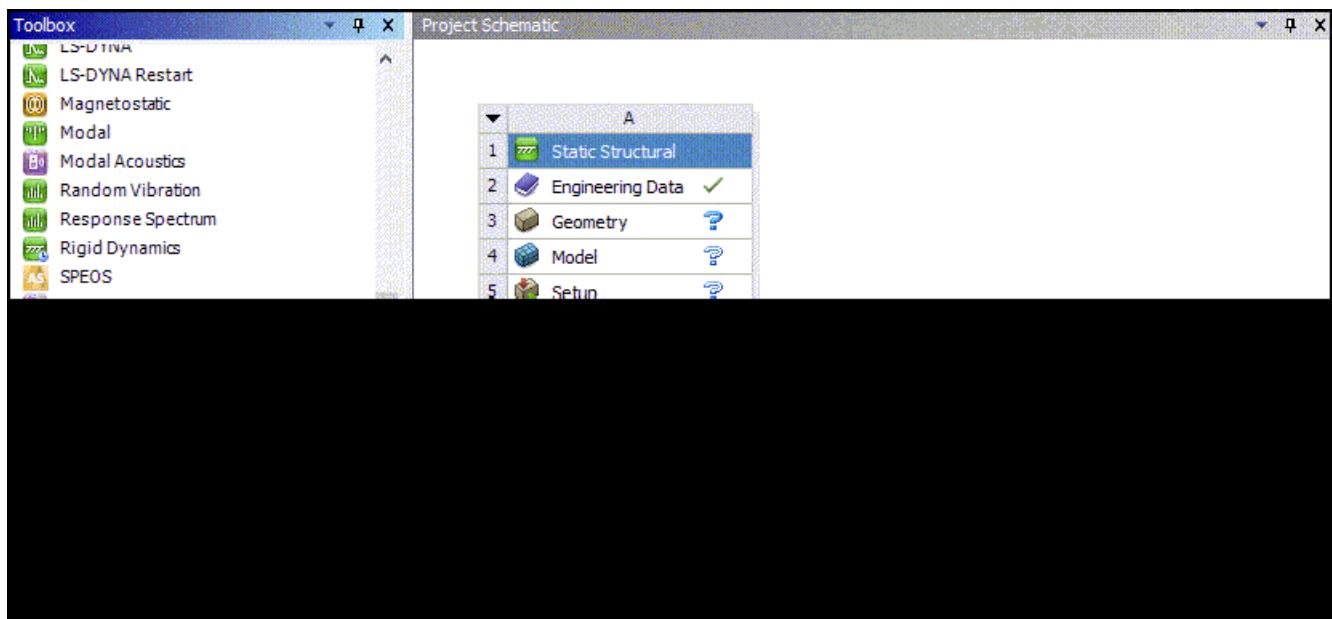
- Topology_Optimization_Example_001.agdb
- Topology_Optimization_Example_001.py

These files are available [here](#).

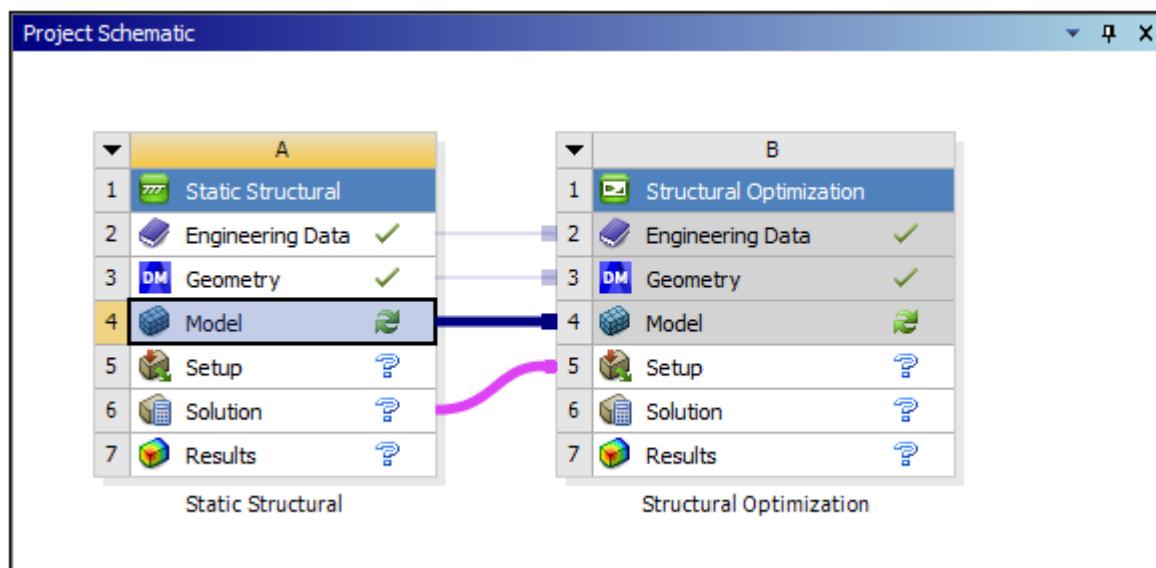
Procedure

The following procedure assumes you understand how to add a system in Workbench and make changes to system cells and properties as well as saving the support files to a known location.

1. Add a **Static Structural** analysis to the **Project Schematic**. Drag and drop a **Structural Optimization** system onto the static system as shown.



- Right-click the **Geometry** cell of the static system and select **Import Geometry > Browse** and select the Topology_Optimization_Example_001.agdb file.



- Right-click the **Model** cell and select **Edit**. This action launches the Mechanical application.
- In Mechanical, select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.
- Select the **Open Script** option (Editor toolbar icon) from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select Topology_Optimization_Example_001.py.
- Select the **Run Script** option (Editor toolbar icon) from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
#Scenario 1 Store all main tree objects as variables
GEOMETRY = Model.Geometry
MESH = Model.Mesh
NAMED_SELECTIONS = Model.NamedSelections
CONNECTIONS = Model.Connections
COORDINATE_SYSTEMS = Model.CoordinateSystems
GLOBAL_COORDINATE_SYSTEM = COORDINATE_SYSTEMS.Children[0]

#Scenario 2 Select Unit System
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

#Scenario 3 Store Named selections as variables
Body_1 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Body_1"]
Body_2 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Body_2"]
Body_3 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Body_3"]
Body_4 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Body_4"]

Optimization_Region_1 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Optimization_Region_1"]
Optimization_Region_2 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Optimization_Region_2"]

Force_Face = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Force_Face"]
Fixed_Face_1 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Fixed_Face_1"]
Fixed_Face_2 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Fixed_Face_2"]

Exclusion_1 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Exclusion_1"]
Exclusion_2 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Exclusion_2"]

#Scenario 4 Add mesh method
MESH.ElementSize = Quantity('0.005 [m]')

MESH_METHOD1 = MESH.AddAutomaticMethod()
MESH_METHOD1.Location = Body_1
MESH_METHOD1.Method = MethodType.Sweep
MESH_METHOD1.Algorithm = MeshMethodAlgorithm.Axisymmetric

#Scenario 5 Add local coordinate systems
COORDINATE_SYSTEM1 = COORDINATE_SYSTEMS.AddCoordinateSystem()
COORDINATE_SYSTEM1.OriginLocation = Body_2

COORDINATE_SYSTEM2 = COORDINATE_SYSTEMS.AddCoordinateSystem()
COORDINATE_SYSTEM2.OriginLocation = Body_4

#Scenario 6 Setup Static Structural analysis
STATIC_STRUCTURAL = Model.Analyses[0]

FIXED_SUPPORT1 = STATIC_STRUCTURAL.AddFixedSupport()
FIXED_SUPPORT1.Location = Fixed_Face_1

FIXED_SUPPORT2 = STATIC_STRUCTURAL.AddFixedSupport()
FIXED_SUPPORT2.Location = Fixed_Face_2

FORCE1 = STATIC_STRUCTURAL.AddForce()
FORCE1.Location = Force_Face
FORCE1.DefineBy = LoadDefineBy.Components
FORCE1.YComponent.Output.DiscreteValues = [Quantity("5000 [N]")]
FORCE1.ZComponent.Output.DiscreteValues = [Quantity("1000 [N]")]

SOLUTION1 = STATIC_STRUCTURAL.Solution
TOTAL_DEFORMATION = SOLUTION1.AddTotalDeformation()
EQUIVALENT_STRESS = SOLUTION1.AddEquivalentStress()
```

```

#Scenario 7 Setup Topology Optimization analysis
TOPOLOGY_OPTIMIZATION = Model.Analyses[1]

OPTIMIZATION_REGION1 = TOPOLOGY_OPTIMIZATION.Children[1]
OPTIMIZATION_REGION1.DesignRegionLocation = Optimization_Region_1
OPTIMIZATION_REGION1.BoundaryCondition = BoundaryConditionType.AllLoadsAndSupports
OPTIMIZATION_REGION1.OptimizationType = OptimizationType.TopologyDensity

# Insert Exclusion Region 1
EXCLUSION_REGION1 = OPTIMIZATION_REGION1.AddExclusionRegion()
EXCLUSION_REGION1.ExclusionRegionLocation = Exclusion_1

# Insert Optimization Region 2
OPTIMIZATION_REGION2 = TOPOLOGY_OPTIMIZATION.AddOptimizationRegion()
OPTIMIZATION_REGION2.DesignRegionLocation = Optimization_Region_2
OPTIMIZATION_REGION2.BoundaryCondition = BoundaryConditionType.AllLoadsAndSupports
OPTIMIZATION_REGION2.OptimizationType = OptimizationType.TopologyDensity

# Insert Exclusion Region 2
EXCLUSION_REGION2 = OPTIMIZATION_REGION2.AddExclusionRegion()
EXCLUSION_REGION2.ExclusionRegionLocation = Exclusion_2

# Mass Constraint
RESPONSE_CONSTRAINT1 = TOPOLOGY_OPTIMIZATION.Children[4]
RESPONSE_CONSTRAINT1.Location = OPTIMIZATION_REGION1
RESPONSE_CONSTRAINT1.PercentageToRetain = 40

# Insert Volume Constraint
RESPONSE_CONSTRAINT2 = TOPOLOGY_OPTIMIZATION.AddVolumeConstraint()
RESPONSE_CONSTRAINT2.Location = OPTIMIZATION_REGION2
RESPONSE_CONSTRAINT2.DefineBy = ResponseConstraintDefineBy.Range
RESPONSE_CONSTRAINT2.PercentageToRetainMin = 40
RESPONSE_CONSTRAINT2.PercentageToRetainMax = 50

# Insert Member Size Constraint
MANUFACTURING_CONSTRAINT1 = TOPOLOGY_OPTIMIZATION.AddMemberSizeManufacturingConstraint()
MANUFACTURING_CONSTRAINT1.Location = OPTIMIZATION_REGION1
MANUFACTURING_CONSTRAINT1.Minimum = ManuMemberSizeControlledType.Manual
MANUFACTURING_CONSTRAINT1.MinSize = Quantity('0.0125 [m]')

# Insert Symmetry Constraints
MANUFACTURING_CONSTRAINT2 = TOPOLOGY_OPTIMIZATION.AddSymmetryManufacturingConstraint()
MANUFACTURING_CONSTRAINT2.Location = Body_2
MANUFACTURING_CONSTRAINT2.CoordinateSystem = COORDINATE_SYSTEM1
MANUFACTURING_CONSTRAINT2.Axis = CoordinateSystemAxisType.PositiveYAxis

MANUFACTURING_CONSTRAINT3 = TOPOLOGY_OPTIMIZATION.AddSymmetryManufacturingConstraint()
MANUFACTURING_CONSTRAINT3.Location = Body_2
MANUFACTURING_CONSTRAINT3.CoordinateSystem = COORDINATE_SYSTEM1
MANUFACTURING_CONSTRAINT3.Axis = CoordinateSystemAxisType.PositiveZAxis

MANUFACTURING_CONSTRAINT4 = TOPOLOGY_OPTIMIZATION.AddSymmetryManufacturingConstraint()
MANUFACTURING_CONSTRAINT4.Location = Body_3
MANUFACTURING_CONSTRAINT4.CoordinateSystem = GLOBAL_COORDINATE_SYSTEM
MANUFACTURING_CONSTRAINT4.Axis = CoordinateSystemAxisType.PositiveZAxis

MANUFACTURING_CONSTRAINT5 = TOPOLOGY_OPTIMIZATION.AddSymmetryManufacturingConstraint()
MANUFACTURING_CONSTRAINT5.Location = Body_3
MANUFACTURING_CONSTRAINT5.CoordinateSystem = GLOBAL_COORDINATE_SYSTEM
MANUFACTURING_CONSTRAINT5.Axis = CoordinateSystemAxisType.PositiveYAxis

MANUFACTURING_CONSTRAINT6 = TOPOLOGY_OPTIMIZATION.AddSymmetryManufacturingConstraint()
MANUFACTURING_CONSTRAINT6.Location = Body_4
MANUFACTURING_CONSTRAINT6.CoordinateSystem = COORDINATE_SYSTEM2
MANUFACTURING_CONSTRAINT6.Axis = CoordinateSystemAxisType.PositiveXAxis

MANUFACTURING_CONSTRAINT7 = TOPOLOGY_OPTIMIZATION.AddSymmetryManufacturingConstraint()
MANUFACTURING_CONSTRAINT7.Location = Body_4
MANUFACTURING_CONSTRAINT7.CoordinateSystem = COORDINATE_SYSTEM2

```

```

MANUFACTURING_CONSTRAINT7.Axis = CoordinateSystemAxisType.PositiveZAxis

# Insert Cyclic Constraints
MANUFACTURING_CONSTRAINT8 = TOPOLOGY_OPTIMIZATION.AddCyclicManufacturingConstraint()
MANUFACTURING_CONSTRAINT8.Location = OPTIMIZATION_REGION1
MANUFACTURING_CONSTRAINT8.NumberofSectors = 4
MANUFACTURING_CONSTRAINT8.CoordinateSystem = GLOBAL_COORDINATE_SYSTEM
MANUFACTURING_CONSTRAINT8.Axis = CoordinateSystemAxisType.PositiveZAxis

# Get Tracker and Result
SOLUTION2 = TOPOLOGY_OPTIMIZATION.Solution
SOLUTION_INFORMATION2 = SOLUTION2.SolutionInformation
TOPOLOGY_DENSITY_TRACKER1 = SOLUTION_INFORMATION2.Children[0]
TOPOLOGY_DENSITY_TRACKER1.ScopingMethod = GeometryDefineByType.AllOptimizationRegions

TOPOLOGY_DENSITY1 = SOLUTION2.Children[1]
TOPOLOGY_DENSITY1.ScopingMethod = GeometryDefineByType.AllOptimizationRegions

# Insert Smoothing
SMOOTHING1 = TOPOLOGY_DENSITY1.AddSmoothing()

#Scenario 8 Solve and Review Results

SOLUTION2.Solve(True)

TOPO_DENS1_OV = TOPOLOGY_DENSITY1.OriginalVolume.Value
TOPO_DENS1_FV = TOPOLOGY_DENSITY1.FinalVolume.Value
TOPO_DENS1_PVO = TOPOLOGY_DENSITY1.PercentVolumeOfOriginal

TOPO_DENS1_OM = TOPOLOGY_DENSITY1.OriginalMass.Value
TOPO_DENS1_FM = TOPOLOGY_DENSITY1.FinalMass.Value
TOPO_DENS1_PMO = TOPOLOGY_DENSITY1.PercentMassOfOriginal

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Structural Optimization (Level Set Based) Analysis

In this example, using the support files, you perform a **Structural Optimization** analysis in Mechanical using a sequence of python journal commands that will define and solve the analysis. This analysis uses the **Topology Optimization - Level Set Based** method.

This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

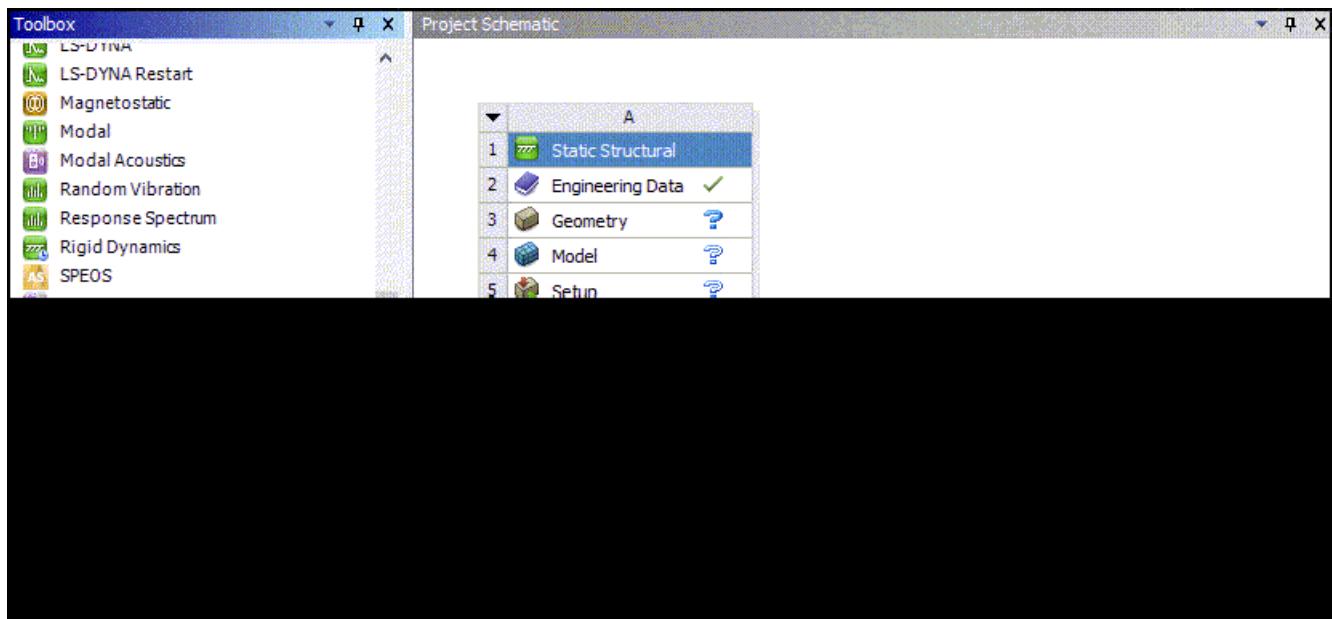
- Structural_Optimization_Level_Set.agdb
- Structural_Optimization_Level_Set.py

These files are available [here](#).

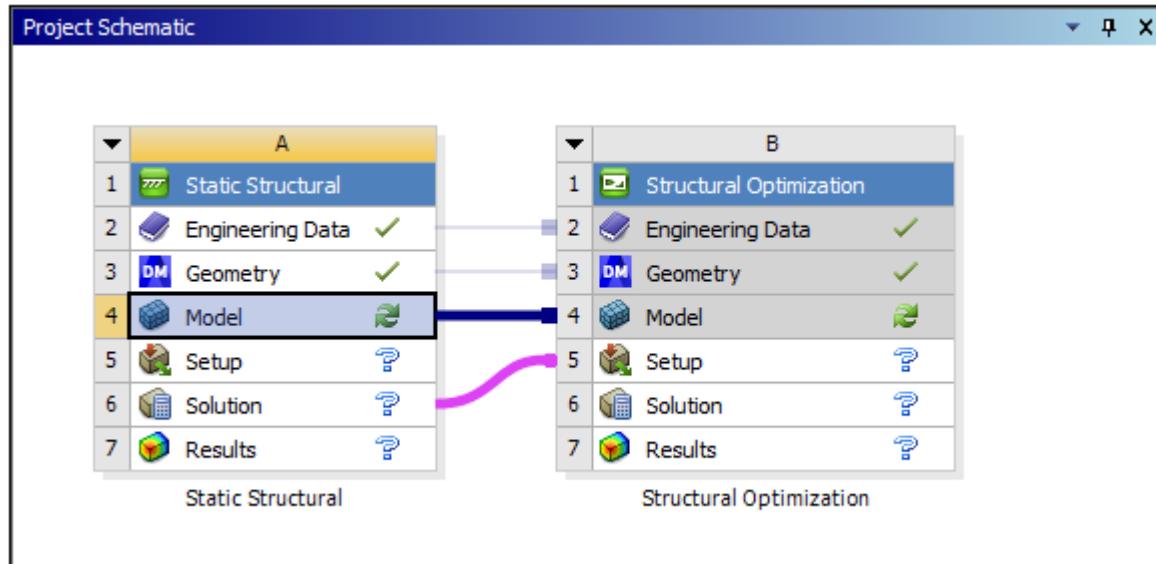
Procedure

The following procedure assumes you understand how to add a system in Workbench and make changes to system cells and properties as well as saving the support files to a known location.

- Add a **Static Structural** analysis to the **Project Schematic**. Drag and drop a **Structural Optimization** system onto the static system as shown.



- Right-click the **Geometry** cell of the static system and select **Import Geometry > Browse** and select the `Structural_Optimization_Level_Set.agdb` file.



- Right-click the **Model** cell and select **Edit**. This action launches the Mechanical application.
- In Mechanical, select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.
- Select the **Open Script** option (highlighted with a red box) from the **Editor** (p. 4) toolbar. Navigate to the proper folder location and select `Structural_Optimization_Level_Set.py`.

6. Select the **Run Script** option ( from the **Editor** (p. 4) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
#Scenario 1 Store all main tree nodes as variables
GEOMETRY = Model.Geometry
MESH = Model.Mesh
NAMED_SELECTIONS = Model.NamedSelections
CONNECTIONS = Model.Connections
COORDINATE_SYSTEMS = Model.CoordinateSystems
GLOBAL_COORDINATE_SYSTEM = COORDINATE_SYSTEMS.Children[0]

#Scenario 2 Select Unit System
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

#Scenario 3 Store Named selections as variable
Fixed_Support_1 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if
Fixed_Support_2 = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if
Force = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name ==

#Scenario 4 Add mesh method
MESH.ElementSize = Quantity('0.005 [m]')
MESH.UseAdaptiveSizing = False

#Scenario 5 Setup Static Structural analysis
STATIC_STRUCTURAL = Model.Analyses[0]

FIXED_SUPPORT1 = STATIC_STRUCTURAL.AddFixedSupport()
FIXED_SUPPORT1.Location = Fixed_Support_1

FIXED_SUPPORT2 = STATIC_STRUCTURAL.AddFixedSupport()
FIXED_SUPPORT2.Location = Fixed_Support_2

FORCE1 = STATIC_STRUCTURAL.AddForce()
FORCE1.Location = Force
FORCE1.DefineBy = LoadDefineBy.Components
FORCE1.XComponent.Output.DiscreteValues = [Quantity("-1000 [N]")]

SOLUTION1 = STATIC_STRUCTURAL.Solution
TOTAL_DEFORMATION = SOLUTION1.AddTotalDeformation()
EQUIVALENT_STRESS = SOLUTION1.AddEquivalentStress()

#Scenario 6 Setup Topology Optimization analysis
TOPOLOGY_OPTIMIZATION = Model.Analyses[1]

OPTIMIZATION_REGION1 = TOPOLOGY_OPTIMIZATION.Children[1]
OPTIMIZATION_REGION1.BoundaryCondition = BoundaryConditionType.AllLoadsAndSupports
OPTIMIZATION_REGION1.OptimizationType = OptimizationType.TopologyLevelSet

# Mass Constraint
RESPONSE_CONSTRAINT1 = TOPOLOGY_OPTIMIZATION.Children[3]
RESPONSE_CONSTRAINT1.DefineBy = ResponseConstraintDefineBy.Range
RESPONSE_CONSTRAINT1.PercentageToRetainMin = 50
RESPONSE_CONSTRAINT1.PercentageToRetainMax = 70

# Insert Reaction Force Constraint
RESPONSE_CONSTRAINT2 = TOPOLOGY_OPTIMIZATION.AddReactionForceConstraint()
RESPONSE_CONSTRAINT2.Location = Fixed_Support_1
RESPONSE_CONSTRAINT2.AxisSelection = AxisSelectionType.XAxis
```

```
RESPONSE_CONSTRAINT2.ReactionForceCriteria = ReactionForceCriteriaType.Sum
RESPONSE_CONSTRAINT2.BoundType = TopoBoundType.LowerBound
RESPONSE_CONSTRAINT2.XComponentMax.Output.DiscreteValues=[Quantity ("500 [N]")]

# Insert Reaction Force Constraint
RESPONSE_CONSTRAINT3 = TOPOLOGY_OPTIMIZATION.AddReactionForceConstraint()
RESPONSE_CONSTRAINT3.Location = Fixed_Support_2
RESPONSE_CONSTRAINT3.AxisSelection = AxisSelectionType.XAxis
RESPONSE_CONSTRAINT3.ReactionForceCriteria = ReactionForceCriteriaType.Sum
RESPONSE_CONSTRAINT3.BoundType = TopoBoundType.LowerBound
RESPONSE_CONSTRAINT3.XComponentMax.Output.DiscreteValues=[Quantity ("100 [N]")]

# Get Tracker and Result
SOLUTION2 = TOPOLOGY_OPTIMIZATION.Solution
SOLUTION_INFORMATION2 = SOLUTION2.SolutionInformation
TOPOLOGY_DENSITY_TRACKER1 = SOLUTION_INFORMATION2.Children[0]

TOPOLOGY_DENSITY1 = SOLUTION2.Children[1]

# Insert Smoothing
SMOOTHING1 = TOPOLOGY_DENSITY1.AddSmoothing()

#Scenario 7 Solve and Review Results

SOLUTION2.Solve(True)

TOPO_DENS1_OV = TOPOLOGY_DENSITY1.OriginalVolume.Value
TOPO_DENS1_FV = TOPOLOGY_DENSITY1.FinalVolume.Value
TOPO_DENS1_PVO = TOPOLOGY_DENSITY1.PercentVolumeOfOriginal

TOPO_DENS1_OM = TOPOLOGY_DENSITY1.OriginalMass.Value
TOPO_DENS1_FM = TOPOLOGY_DENSITY1.FinalMass.Value
TOPO_DENS1_PMO = TOPOLOGY_DENSITY1.PercentMassOfOriginal
```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Structural Optimization Lattice Analysis

In this example, using the support files, you perform a **Structural Optimization** analysis in Mechanical using a sequence of python journal commands that will define and solve the analysis. This analysis uses the **Lattice Optimization** method.

This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

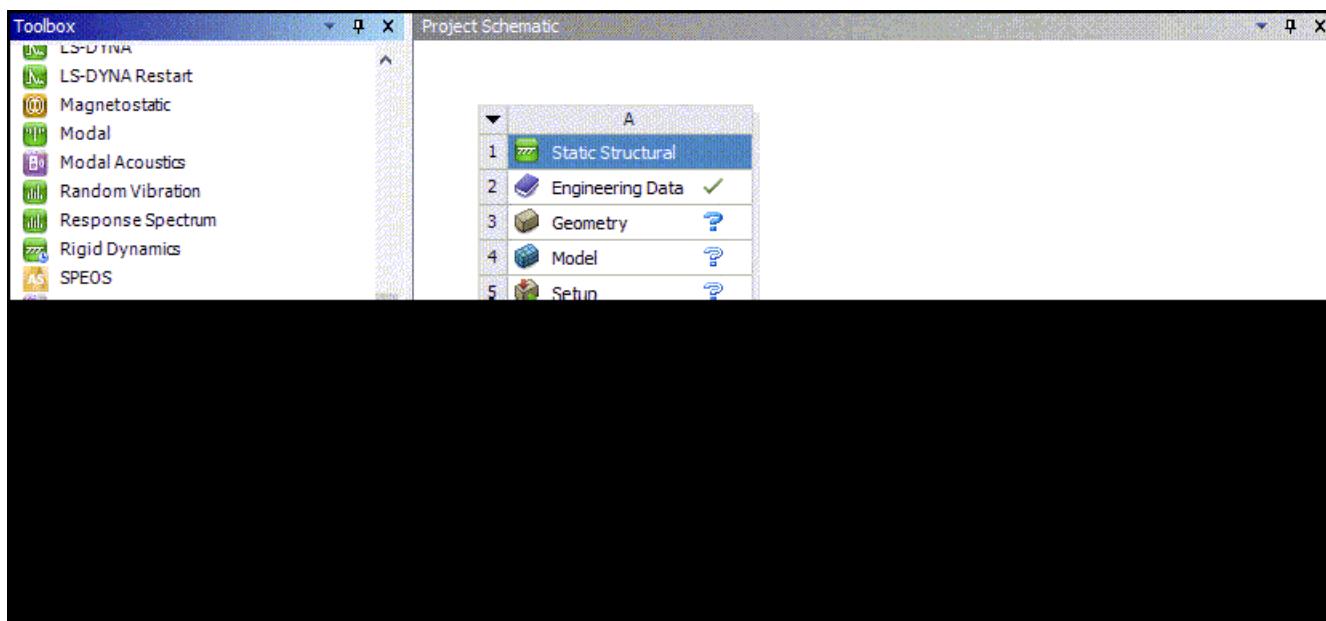
- Structural_Optimization_Lattice.agdb
- Structural_Optimization_Lattice.py

These files are available [here](#).

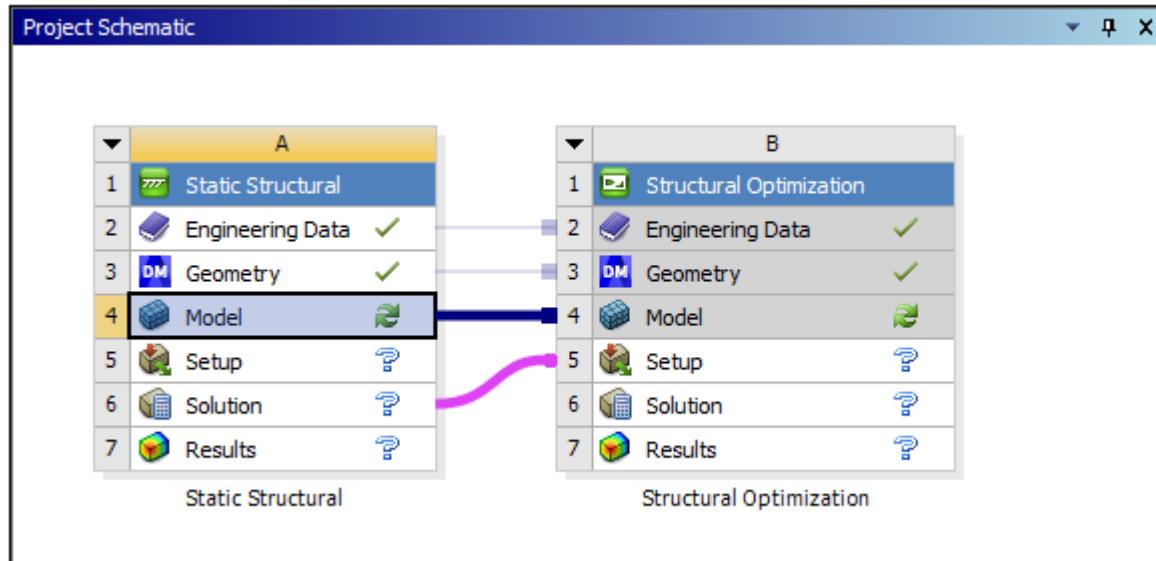
Procedure

The following procedure assumes you understand how to add a system in Workbench and make changes to system cells and properties as well as saving the support files to a known location.

- Add a **Static Structural** analysis to the **Project Schematic**. Drag and drop a **Structural Optimization** system onto the static system as shown.



- Right-click the **Geometry** cell of the static system and select **Import Geometry > Browse** and select the `Structural_Optimization_Lattice.agdb` file.



- Right-click the **Model** cell and select **Edit**. This action launches the Mechanical application.
- In Mechanical, select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.
- Select the **Open Script** option (highlighted with a red box) from the **Editor** (p. 4) toolbar. Navigate to the proper folder location and select `Structural_Optimization_Lattice.py`.

6. Select the **Run Script** option ( from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
#Scenario 1 Store all main tree nodes as variables
GEOMETRY = Model.Geometry
MESH = Model.Mesh
NAMED_SELECTIONS = Model.NamedSelections
CONNECTIONS = Model.Connections
COORDINATE_SYSTEMS = Model.CoordinateSystems
GLOBAL_COORDINATE_SYSTEM = COORDINATE_SYSTEMS.Children[0]

#Scenario 2 Select Unit System
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

#Scenario 3 Store Named selections as variable
Body = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Fixed_Support"]
Pressure = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Pressure"]
Internal_Faces = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Internal_Faces"]
Top_Face = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Top_Face"]

#Scenario 4 Add mesh method
FACE_SIZING = MESH.AddSizing()
FACE_SIZING.Location = Top_Face
FACE_SIZING.ElementSize = Quantity('0.01524 [m]')

MESH_METHOD1 = MESH.AddAutomaticMethod()
MESH_METHOD1.Location = Body
MESH_METHOD1.Method = MethodType.MultiZone

MESH.GenerateMesh()

COORDINATE_SYSTEM1 = COORDINATE_SYSTEMS.AddCoordinateSystem()
COORDINATE_SYSTEM1.OriginLocation = Internal_Faces

#Scenario 5 Setup Static Structural analysis
STATIC_STRUCTURAL = Model.Analyses[0]

FIXED_SUPPORT1 = STATIC_STRUCTURAL.AddFixedSupport()
FIXED_SUPPORT1.Location = Fixed_Support

PRESSURE1 = STATIC_STRUCTURAL.AddPressure()
PRESSURE1.Location = Pressure
PRESSURE1.Magnitude.Output.DiscreteValues = [Quantity("50 [Pa]")]

SOLUTION1 = STATIC_STRUCTURAL.Solution
TOTAL_DEFORMATION = SOLUTION1.AddTotalDeformation()
EQUIVALENT_STRESS = SOLUTION1.AddEquivalentStress()

#Scenario 6 Setup Topology Optimization analysis
TOPOLOGY_OPTIMIZATION = Model.Analyses[1]

OPTIMIZATION_REGION1 = TOPOLOGY_OPTIMIZATION.Children[1]
OPTIMIZATION_REGION1.BoundaryCondition = BoundaryConditionType.AllLoadsAndSupports
OPTIMIZATION_REGION1.OptimizationType = OptimizationType.Lattice

OPTIMIZATION_REGION1.LatticeType = LatticeType.Cubic
OPTIMIZATION_REGION1.LatticeMinDensity = 0.3
OPTIMIZATION_REGION1.LatticeMaxDensity = 0.8
```

```

OPTIMIZATION_REGION1.LatticeSize = Quantity("0.06 [m]")

# Mass Constraint
RESPONSE_CONSTRAINT1 = TOPOLOGY_OPTIMIZATION.Children[3]
RESPONSE_CONSTRAINT1.DefineBy = ResponseConstraintDefineBy.Range
RESPONSE_CONSTRAINT1.PercentageToRetainMin = 30
RESPONSE_CONSTRAINT1.PercentageToRetainMax = 35

# Insert Symmetry Constraint
MANUFACTURING_CONSTRAINT1 = TOPOLOGY_OPTIMIZATION.AddSymmetryManufacturingConstraint()
MANUFACTURING_CONSTRAINT1.Location = Body
MANUFACTURING_CONSTRAINT1.CoordinateSystem = COORDINATE_SYSTEM1
MANUFACTURING_CONSTRAINT1.Axis = CoordinateSystemAxisType.PositiveZAxis

# Insert Global von-Mises Stress Constraint
RESPONSE_CONSTRAINT2 = TOPOLOGY_OPTIMIZATION.AddGlobalVonMisesStressConstraint()
RESPONSE_CONSTRAINT2.Maximum.Output.DiscreteValues=[Quantity ("150 [Pa]")]

# Insert Moment of Inertia Constraint
RESPONSE_CONSTRAINT3 = TOPOLOGY_OPTIMIZATION.AddMomentOfInertiaConstraint()
RESPONSE_CONSTRAINT3.PercentageToRetainMin = 40
RESPONSE_CONSTRAINT3.PercentageToRetainMax = 50

# Get Tracker and Result
SOLUTION2 = TOPOLOGY_OPTIMIZATION.Solution
SOLUTION_INFORMATION2 = SOLUTION2.SolutionInformation
LATTICE_DENSITY_TRACKER1 = SOLUTION_INFORMATION2.Children[0]

LATTICE_DENSITY1 = SOLUTION2.Children[1]

#Scenario 7 Solve and Review Results

SOLUTION2.Solve(True)

LATTICE_DENS1_OV = LATTICE_DENSITY1.OriginalVolume.Value
LATTICE_DENS1_FV = LATTICE_DENSITY1.FinalVolume.Value
LATTICE_DENS1_PVO = LATTICE_DENSITY1.PercentVolumeOfOriginal

LATTICE_DENS1_OM = LATTICE_DENSITY1.OriginalMass.Value
LATTICE_DENS1_FM = LATTICE_DENSITY1.FinalMass.Value
LATTICE_DENS1_PMO = LATTICE_DENSITY1.PercentMassOfOriginal

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Structural Optimization Shape Optimization Analysis

In this example, using the support files, you perform a **Structural Optimization** analysis in Mechanical using a sequence of python journal commands that will define and solve the analysis. This analysis uses the **Shape Optimization** method.

This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

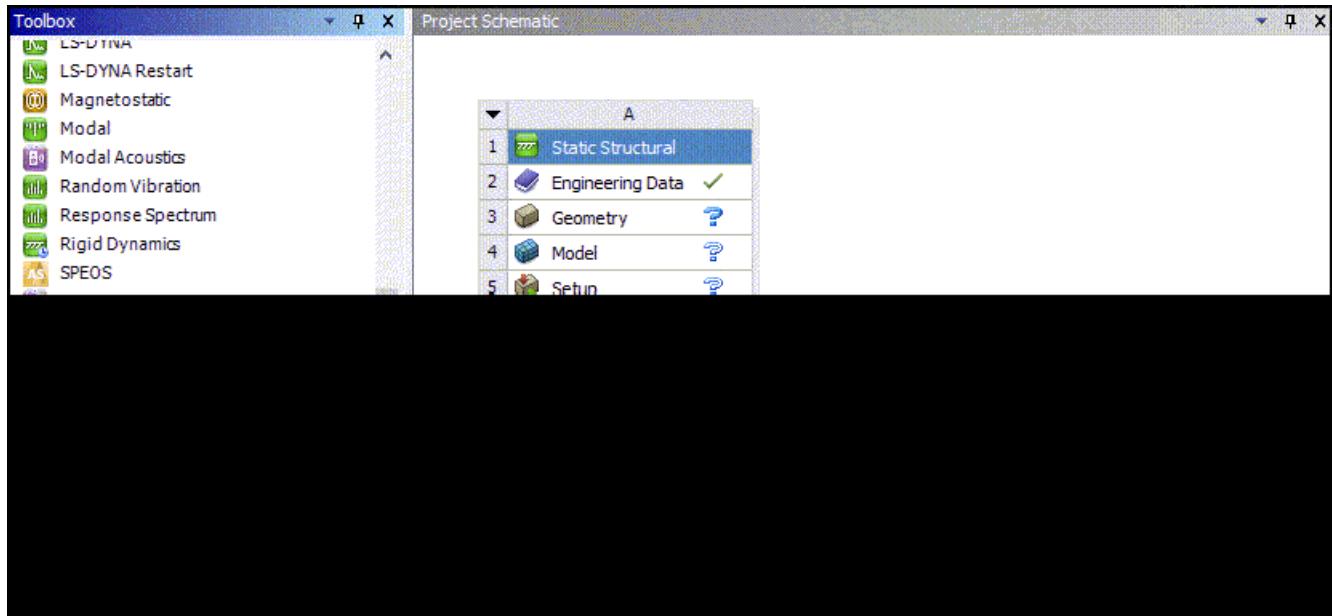
- Structural_Optimization_Shape_Optimization.agdb
- Structural_Optimization_Shape_Optimization.py

These files are available [here](#).

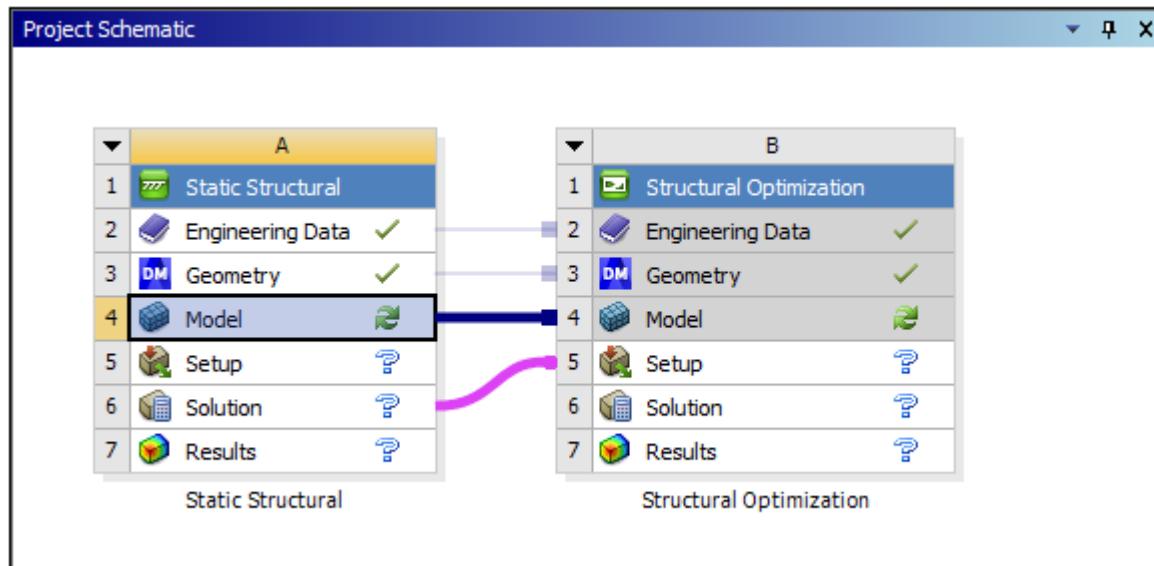
Procedure

The following procedure assumes you understand how to add a system in Workbench and make changes to system cells and properties as well as saving the support files to a known location.

1. Add a **Static Structural** analysis to the **Project Schematic**. Drag and drop a **Structural Optimization** system onto the static system as shown.



2. Right-click the **Geometry** cell of the static system and select **Import Geometry > Browse** and select the `Structural_Optimization_Shape_Optimization.agdb` file.



3. Right-click the **Model** cell and select **Edit**. This action launches the Mechanical application.
4. In Mechanical, select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.

5. Select the **Open Script** option (from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select Structural_Optimization_Shape_Optimization.py.

6. Select the **Run Script** option (from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
#Scenario 1 Store all main tree nodes as variables
GEOMETRY = Model.Geometry
MESH = Model.Mesh
NAMED_SELECTIONS = Model.NamedSelections
CONNECTIONS = Model.Connections
COORDINATE_SYSTEMS = Model.CoordinateSystems
GLOBAL_COORDINATE_SYSTEM = COORDINATE_SYSTEMS.Children[0]

#Scenario 2 Select Unit System
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

#Scenario 3 Store Named selections as variable
Body = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Fixed_Support"]
Force = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Force"]
Exclusion = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.Name == "Exclusion"]

#Scenario 4 Add mesh method
MESH.ElementSize = Quantity('0.005 [m]')
MESH.UseAdaptiveSizing = False

MESH_METHOD1 = MESH.AddAutomaticMethod()
MESH_METHOD1.Location = Body
MESH_METHOD1.Method = MethodType.AllTriAllTet

MESH.GenerateMesh()

#Scenario 5 Setup Static Structural analysis
STATIC_STRUCTURAL = Model.Analyses[0]

FIXED_SUPPORT1 = STATIC_STRUCTURAL.AddFixedSupport()
FIXED_SUPPORT1.Location = Fixed_Support

FORCE1 = STATIC_STRUCTURAL.AddForce()
FORCE1.Location = Force
FORCE1.DefineBy = LoadDefineBy.Components
FORCE1.ZComponent.Output.DiscreteValues = [Quantity("25000 [N]")]

SOLUTION1 = STATIC_STRUCTURAL.Solution
TOTAL_DEFORMATION = SOLUTION1.AddTotalDeformation()
EQUIVALENT_STRESS = SOLUTION1.AddEquivalentStress()

#Scenario 6 Setup Topology Optimization analysis
TOPOLOGY_OPTIMIZATION = Model.Analyses[1]

OPTIMIZATION_REGION1 = TOPOLOGY_OPTIMIZATION.Children[1]
OPTIMIZATION_REGION1.ExclusionRegionLocation = Exclusion
OPTIMIZATION_REGION1.OptimizationType = OptimizationType.Shape

OPTIMIZATION_REGION1.ShapeMoveLimitControl = TopoPropertyControlType.ProgramControlled
OPTIMIZATION_REGION1.MaxCumulatedDisplacementControl = TopoPropertyControlType.ProgramControlled
OPTIMIZATION_REGION1.MeshDeformationToleranceControl = TopoPropertyControlType.ProgramControlled

# Objective - Minimize Volume
```

```
OBJECTIVE = TOPOLOGY_OPTIMIZATION.GetChildren(DataModelObjectCategory.Objective, True)
OBJECTIVE[0].Activate()
OBJECTIVE[0].Worksheet.SetObjectiveType(0, ObjectiveType.MinimizeVolume)

# Mass Constraint
RESPONSE_CONSTRAINT1 = TOPOLOGY_OPTIMIZATION.Children[3]
RESPONSE_CONSTRAINT1.Suppress = True

# Insert Compliance Constraint
RESPONSE_CONSTRAINT2 = TOPOLOGY_OPTIMIZATION.AddComplianceConstraint()
RESPONSE_CONSTRAINT2.ComplianceLimit.Output.DiscreteValues=[Quantity ("0.27 [J]")]

# Get Tracker and Result
SOLUTION2 = TOPOLOGY_OPTIMIZATION.Solution
SOLUTION_INFORMATION2 = SOLUTION2.SolutionInformation
TOPOLOGY_DENSITY_TRACKER1 = SOLUTION_INFORMATION2.Children[0]

TOPOLOGY_DENSITY1 = SOLUTION2.Children[1]

#Scenario 7 Solve and Review Results

SOLUTION2.Solve(True)

TOPO_DENS1_OV = TOPOLOGY_DENSITY1.OriginalVolume.Value
TOPO_DENS1_FV = TOPOLOGY_DENSITY1.FinalVolume.Value
TOPO_DENS1_PVO = TOPOLOGY_DENSITY1.PercentVolumeOfOriginal

TOPO_DENS1_OM = TOPOLOGY_DENSITY1.OriginalMass.Value
TOPO_DENS1_FM = TOPOLOGY_DENSITY1.FinalMass.Value
TOPO_DENS1_PMO = TOPOLOGY_DENSITY1.PercentMassOfOriginal
```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Rigid Dynamics: Contact Specification

In this example, using the support files, you will insert a Rigid Dynamics analysis object into an undefined Mechanical session and execute a sequence of python journal commands that defines an analysis, with a focus on contact, and solves the analysis.

This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

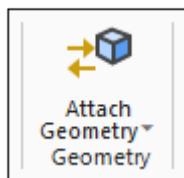
- Mechanical_RBD_Contact_Example.agdb
- Mechanical_RBD_Contact_Example.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Rigid Dynamics** system into the tree.

3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the **Geometry Context tab**. Navigate to the proper folder location and select **Mechanical_RBD_Contact_Example.agdb**.



4. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.

5. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select **Mechanical_RBD_Contact_Example.py**.

6. Select the **Run Script** option () from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Section 1 - Set up the Tree Object Items

connections = Model.Connections
transient_rbd = Model.Analyses[0]
analysis_settings = transient_rbd.Children[0]
mesh = Model.Mesh
solution = transient_rbd.Solution

# Section 2 - Define the Named Selections

ns_torus_surface = DataModel.GetObjectsByName("NS_torus_surface")[0]
ns_cone_surface = DataModel.GetObjectsByName("NS_cone_surface")[0]
ns_fixed_surface = DataModel.GetObjectsByName("NS_fixed_surface")[0]

# Section 3 - Define the Contact

contact = connections.AddContactRegion()
contact.SourceLocation = ns_torus_surface
contact.TargetLocation = ns_cone_surface
contact.ContactType = ContactType.Frictionless
contact.PinballRegion = ContactPinballType.ProgramControlled
contact.RestitutionFactor = 0.5
contact.RBDCContactDetection = DSRBDCContactDetection.kCDGeometryBased

# Section 4 - Define the Fixed Joint

fixed_joint = connections.AddJoint()
fixed_joint.ConnectionType=JointScopingType.BodyToGround
fixed_joint.Type = JointType.Fixed
fixed_joint.MobileLocation = ns_fixed_surface

# Section 5 - Define the Mesh

mesh.PhysicsPreference = MeshPhysicsPreferenceType.Mechanical
mesh.ElementOrder=ElementOrder.Quadratic
mesh.ElementSize = Quantity("0.005 [m]")
mesh.Resolution = 4
mesh.GenerateMesh()
```

```

# Section 6 - Insert Standard Earth Gravity

gravity = transient_rbd.AddEarthGravity()
gravity.Direction=GravityOrientationType.NegativeZAxis

# Section 7 - Define the Analysis Settings

analysis_settings.SetStepEndTime( 1, Quantity("0.4 [s]"))
analysis_settings.SetAutomaticTimeStepping(1, AutomaticTimeStepping.On)
analysis_settings.SetInitialTimeStep(1,Quantity("0.0001 [s]"))
analysis_settings.SetMinimumTimeStep(1,Quantity("0.0000001 [s]"))
analysis_settings.SetMaximumTimeStep(1,Quantity("0.01 [s]"))

# Section 8 - Define the Contact Results

total_contact_force_reaction = solution.AddForceReaction()
total_contact_force_reaction.LocationMethod=LocationDefinitionMethod.ContactRegion
total_contact_force_reaction.ContactRegionSelection = contact
total_contact_force_reaction.ContactForce=ContactForceType.Total

tangent_contact_force_reaction = solution.AddForceReaction()
tangent_contact_force_reaction.LocationMethod=LocationDefinitionMethod.ContactRegion
tangent_contact_force_reaction.ContactRegionSelection = contact
tangent_contact_force_reaction.ContactForce=ContactForceType.Tangent

normal_contact_force_reaction = solution.AddForceReaction()
normal_contact_force_reaction.LocationMethod=LocationDefinitionMethod.ContactRegion
normal_contact_force_reaction.ContactRegionSelection = contact
normal_contact_force_reaction.ContactForce=ContactForceType.Normal

# Section 9 - Set up the Unit Systems

ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS
ExtAPI.Application.ActiveAngleUnit = AngleUnitType.Radian
ExtAPI.Application.ActiveAngularVelocityUnit=AngularVelocityUnitType.RadianPerSecond

# Section 10 - Solve the System and Define the Contact Results

solution.Solve(True)

total_contact_force_reaction_z = total_contact_force_reaction.ZAxis.Value
total_contact_force_reaction_maximum_z = total_contact_force_reaction.MaximumZAxis.Value
tangent_contact_force_reaction_maximum_total = total_contact_force_reaction.MaximumTotal.Value
normal_contact_force_reaction_maximum_total = normal_contact_force_reaction.MaximumTotal.Value
normal_contact_force_reaction_minimum_z = normal_contact_force_reaction.MinimumZAxis.Value

# Section 11 - Insert and Evaluate the Energy Probe Results

energy_probe = solution.AddEnergyProbe()
solution.EvaluateAllResults()

maximum_potential_energy = energy_probe.MaximumPotentialEnergy.Value
maximum_kinetic_energy = energy_probe.MaximumKineticEnergy.Value
maximum_total_energy = energy_probe.MaximumTotalEnergy.Value
maximum_external_energy = energy_probe.MaximumExternalEnergy.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Rigid Dynamics: Flexible Part Specification

In this example, using the support files, you will insert a Rigid Dynamics analysis object into an undefined Mechanical session and execute a sequence of python journal commands that defines an analysis, with a focus on flexible part definition, and solves the analysis.

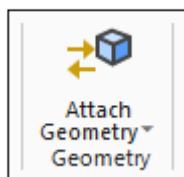
This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Mechanical_RBD_Flexible_Part_Example.agdb
- Mechanical_RBD_Flexible_Part_Example.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Rigid Dynamics** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the [Geometry Context tab](#). Navigate to the proper folder location and select `Mechanical_RBD_Flexible_Part_Example.agdb`.



4. Select the [Automation tab](#) and select the **Scripting** option to open the **Mechanical Scripting** pane.

5. Select the **Open Script** option (from the [Editor \(p. 4\)](#) toolbar. Navigate to the proper folder location and select `Mechanical_RBD_Flexible_Part_Example.py`.

6. Select the **Run Script** option (from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Section 1 - Set up the Tree Object Items

geometry = Model.Geometry
connections = Model.Connections
transient_rbd = Model.Analyses[0]
analysis_settings = transient_rbd.Children[0]
solution = transient_rbd.Solution

# Section 2 - Define the Named Selections

ns_face1 = DataModel.GetObjectsByName("NS_FACE1")[0]
ns_face2 = DataModel.GetObjectsByName("NS_FACE2")[0]
ns_cms1 = DataModel.GetObjectsByName("NS_CMS1")[0]

# Section 3 - Define the Flexible Cms1 Part and Suppress Solid

cms1 = geometry.Children[0].Children[0]
```

```

solid = geometry.Children[1].Children[0]
cms1.StiffnessBehavior = StiffnessBehavior.Flexible
solid.Suppressed = True

# Section 4 - Set up the General Revolute Joint

general_revolute_joint = connections.AddJoint()
general_revolute_joint.ConnectionType=JointScopingType.BodyToGround
general_revolute_joint.Type = JointType.General
general_revolute_joint.MobileLocation = ns_face1
general_revolute_joint.Rotations = JointRotationDOFType.FreeX

# Section 5 - Set up the General Joint with 6 DOF

general_all_free = connections.AddJoint()
general_all_free.ConnectionType=JointScopingType.BodyToGround
general_all_free.Type = JointType.General
general_all_free.TranslationX=FixedOrFree.Free
general_all_free.TranslationY=FixedOrFree.Free
general_all_free.TranslationZ=FixedOrFree.Free
general_all_free.Rotations=JointRotationDOFType.FreeAll
general_all_free.MobileLocation = ns_face2

# Section 6 - Generate Condensed Geometries

condensed_geometry = Model.AddCondensedGeometry()
condensed_part = condensed_geometry.AddCondensedPart()
condensed_part.GeometrySelection = ns_cms1
condensed_part.NumberOfModes = 10
condensed_part.DetectCondensedPartInterface()
condensed_part.GenerateCondensedParts()

# Section 7 - Define the Analysis Settings

analysis_settings.SetStepEndTime( 1, Quantity("0.5 [s]"))
analysis_settings.SetAutomaticTimeStepping( 1, AutomaticTimeStepping.On)
analysis_settings.SetInitialTimeStep( 1, Quantity("1e-4[s]"))
analysis_settings.SetMinimumTimeStep( 1, Quantity("1e-7[s]"))
analysis_settings.SetMaximumTimeStep( 1, Quantity("1e-3[s]"))
analysis_settings.SetStoreResultAt( 1, TimePointsOptions.EquallySpacedPoints)
analysis_settings.SetStoreResultAtValue( 1, 200)

# Section 8 - Insert Standard Earth Gravity

gravity = transient_rbd.AddEarthGravity()
gravity.Direction=GravityOrientationType.PositiveYAxis

# Section 9 - Define the Expansion Settings

expansion_settings = solution.Children[1]
expansion_settings.Activate()
expansion_settings_state = expansion_settings.GetExpansionState( condensed_part)
expansion_settings_state.Stress = True
expansion_settings_state.Displacement = True
expansion_settings.SetExpansionState( condensed_part, expansion_settings_state)

# Section 10 - Insert the Condensed Part Results

total_deformation = solution.AddTotalDeformation()

joint_probe_deformation = solution.AddJointProbe()
joint_probe_deformation.BoundaryConditionSelection = general_all_free
joint_probe_deformation.ResultType = ProbeResultType.DeformationProbe

equivalent_stress = solution.AddEquivalentStress()

middle_principal_strain = solution.AddMiddlePrincipalElasticStrain()

# Section 11 - Set up the Unit Systems

ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

```

```

ExtAPI.Application.ActiveAngleUnit = AngleUnitType.Radian
ExtAPI.Application.ActiveAngularVelocityUnit=AngularVelocityUnitType.RadianPerSecond

# Section 12 - Solve the System and Define the Condensed Part Results

solution.Solve(True)

total_deformation_maximum = total_deformation.Maximum.Value
equivalent_stress_maximum = equivalent_stress.Maximum.Value
middle_principal_stress_maximum = middle_principal_strain.Maximum.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Rigid Dynamics: General Analysis

In this example, using the support files, you will insert a Rigid Dynamics analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will define and solve the analysis.

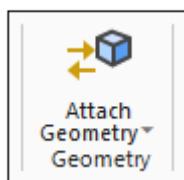
This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Mechanical_RBD_Geometry.agdb
- Mechanical_RBD_Script.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Rigid Dynamics** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the **Geometry Context Tab**. Navigate to the proper folder location and select **Mechanical_RBD_Geometry.agdb**.



4. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.

5. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select **Mechanical_RBD_Script.py**.

6. Select the **Run Script** option ( from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Scenario 1 - Set up the Tree Object Items

connections = Model.Connections
mesh = Model.Mesh
transient_rbd = Model.Analyses[0]
analysis_settings = transient_rbd.Children[0]
solution = transient_rbd.Solution

# Scenario 2 - Define Named Selections

ns_sphere_surface = DataModel.GetObjectsByName("NS_sphere_surface")[0]
ns_concave_surface = DataModel.GetObjectsByName("NS_concave_surface")[0]
ns_fixed_surface = DataModel.GetObjectsByName("NS_fixed_surface")[0]

# Scenario 3 - Define Fixed Joint

fixed_joint = connections.AddJoint()
fixed_joint.ConnectionType = JointScopingType.BodyToGround
fixed_joint.Type = JointType.Fixed
fixed_joint.MobileLocation = ns_fixed_surface

# Scenario 4 - Define Contact

contact = connections.AddContactRegion()
contact.SourceLocation = ns_sphere_surface
contact.TargetLocation = ns_concave_surface
contact.RestitutionFactor = 0

# Scenario 5 - Define Mesh Settings

mesh.Resolution = 4
mesh.ElementSize = Quantity("0.005 [m]")
mesh.GenerateMesh()

# Scenario 6 - Define Analysis Settings

analysis_settings.NumberOfSteps = 1
analysis_settings.SetStepEndTime(1, Quantity("0.5 [s]"))
analysis_settings.SetAutomaticTimeStepping(1, AutomaticTimeStepping.On)
analysis_settings.SetInitialTimeStep(1, Quantity("0.0001 [s]"))
analysis_settings.SetMinimumTimeStep(1, Quantity("0.0000001 [s]"))
analysis_settings.SetMaximumTimeStep(1, Quantity("0.05 [s]"))

# Scenario 7 - Insert Standard Earth Gravity

gravity = transient_rbd.AddEarthGravity()
gravity.Direction = GravityOrientationType.NegativeYAxis

# Scenario 8 - Insert Results

total_deformation = solution.AddTotalDeformation()

joint_probe_total_moment = solution.AddJointProbe()
joint_probe_total_moment.BoundaryConditionSelection = fixed_joint
joint_probe_total_moment.ResultType = ProbeResultType.MomentReaction
joint_probe_total_moment.ResultSelection = ProbeDisplayFilter.XAxis

# Scenario 9 - Set up Unit System

ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS
ExtAPI.Application.ActiveAngleUnit = AngleUnitType.Radian
```

```

ExtAPI.Application.ActiveAngularVelocityUnit=AngularVelocityUnitType.RadianPerSecond

# Scenario 10 - Solve and Define the Results

solution.Solve(True)

maximum_total_deformation = total_deformation.Maximum.Value
minimum_total_deformation = total_deformation.Minimum.Value
maximum_of_maximum_total_deformation = total_deformation.MaximumOfMaximumOverTime.Value
minimum_of_minimum_total_deformation = total_deformation.MinimumOfMaximumOverTime.Value
maximum_x_total_moment = joint_probe_total_moment.MaximumXAxis.Value
minimum_x_total_moment = joint_probe_total_moment.MinimumXAxis.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Rigid Dynamics: Joint Specification

In this example, using the support files, you will insert a Rigid Dynamics analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will specify a joint type and all of its required definitions, and results, and then solve the analysis.

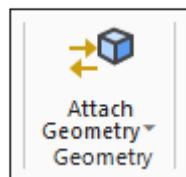
This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Mechanical_RBD_Joint_Example.agdb
- Mechanical_RBD_Joint_Example.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Rigid Dynamics** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the [Geometry Context tab](#). Navigate to the proper folder location and select `Mechanical_RBD_Joint_Example.agdb`.



4. Select the **Automation** tab and select the **Scripting** option to open the **Mechanical Scripting** pane.

5. Select the **Open Script** option () from the [Editor \(p. 4\)](#) toolbar. Navigate to the proper folder location and select Mechanical_RBD_Joint_Example.py.
6. Select the **Run Script** option () from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Section 1 - Set up the Tree Object Items

connections = Model.Connections
transient_rbd = Model.Analyses[0]
analysis_settings = transient_rbd.Children[0]
solution = transient_rbd.Solution

# Section 2 - Define the Named Selections

ns_body_1 = DataModel.GetObjectsByName("NS_Body_1")[0]
ns_body_2 = DataModel.GetObjectsByName("NS_Body_2")[0]
ns_face_1 = DataModel.GetObjectsByName("NS_Face_1")[0]
ns_face_2 = DataModel.GetObjectsByName("NS_Face_2")[0]
ns_face_3 = DataModel.GetObjectsByName("NS_Face_3")[0]
ns_face_4 = DataModel.GetObjectsByName("NS_Face_4")[0]
ns_face_5 = DataModel.GetObjectsByName("NS_Face_5")[0]

# Section 3 - Delete the contact

contact = connections.Children[0].Children[0]
contact.Delete()

# Section 4 - Define the Fixed Joint

fixed_joint = connections.AddJoint()
fixed_joint.ConnectionType=JointScopingType.BodyToGround
fixed_joint.Type = JointType.Fixed
fixed_joint.MobileLocation = ns_face_1

# Section 5 - Define the General Joint

general_joint = connections.AddJoint()
general_joint.ConnectionType=JointScopingType.BodyToBody
general_joint.Type = JointType.General
general_joint.ReferenceLocation = ns_face_2
general_joint.MobileLocation = ns_face_3

# Section 6 - Define the DOF of the General Joint

general_joint.TranslationX=FixedOrFree.Free
general_joint.TranslationY=FixedOrFree.Fixed
general_joint.TranslationZ=FixedOrFree.Fixed
general_joint.Rotations=JointRotationDOFType.FreeY

# Section 7 - Define the Worksheet Properties of the General Joint

worksheet = general_joint.BushingWorksheet
worksheet.SetBushingStiffnessPerUnitX(0, 200)
worksheet.SetBushingDampingPerUnitX(0, 10)

# Section 8 - Define the Frictional Parameters of the General Joint

general_joint.FrictionCoefficient = 0.1
general_joint.Radius = Quantity("1e-2[m]")

# Section 9 - Orientate the General Joint Reference Coordinate System
```

```

reference_coordinate_system = general_joint.ReferenceCoordinateSystem
reference_coordinate_system.PrimaryAxis=CoordinateSystemAxisType.PositiveXAxis
reference_coordinate_system.PrimaryAxisDefineBy=CoordinateSystemAlignmentType.GlobalX
reference_coordinate_system.SecondaryAxis=CoordinateSystemAxisType.PositiveZAxis
reference_coordinate_system.SecondaryAxisDefineBy=CoordinateSystemAlignmentType.GlobalZ

# Section 10 - Insert Standard Earth Gravity

gravity = transient_rbd.AddEarthGravity()
gravity.Direction=GravityOrientationType.PositiveXAxis

# Section 11 - Define the Joint Probe Results

joint_total_force = solution.AddJointProbe()
joint_total_force.BoundaryConditionSelection = general_joint
joint_total_force.ResultType=ProbeResultType.ForceReaction

joint_elastic_force = solution.AddJointProbe()
joint_elastic_force.BoundaryConditionSelection = general_joint
joint_elastic_force.ResultType=ProbeResultType.ElasticForce

joint_damping_force = solution.AddJointProbe()
joint_damping_force.BoundaryConditionSelection = general_joint
joint_damping_force.ResultType=ProbeResultType.DampingForce

joint_relative_velocity = solution.AddJointProbe()
joint_relative_velocity.BoundaryConditionSelection = general_joint
joint_relative_velocity.ResultType=ProbeResultType.VelocityProbe
joint_relative_velocity.ResultSelection=ProbeDisplayFilter.XAxis

# Section 12 - Set up the Unit Systems

ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS
ExtAPI.Application.ActiveAngleUnit = AngleUnitType.Radian
ExtAPI.Application.ActiveAngularVelocityUnit=AngularVelocityUnitType.RadianPerSecond

# Section 13 - Solve the System and Define the Results

solution.Solve(True)

joint_total_force_maximum = joint_total_force.MaximumTotal.Value
joint_elastic_force_final_total = joint_elastic_force.Total.Value
joint_elastic_force_final_x_axis = joint_elastic_force.XAxis.Value
joint_damping_force_maximum_total = joint_damping_force.MaximumTotal.Value
joint_relative_velocity_maximum_x = joint_relative_velocity.MaximumXAxis.Value
joint_elastic_force_maximum_total = joint_elastic_force.MaximumTotal.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Rigid Dynamics: Variable Load Specification

In this example, using the support files, you will insert a Rigid Dynamics analysis object into an undefined Mechanical session and execute a sequence of python journal commands that defines an analysis, with a focus on the VariableLoad Extension, and solves the analysis.

This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

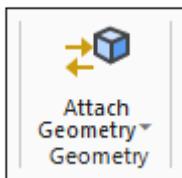
- Mechanical_RBD_Variable_Load_Example.agdb

- Mechanical_RBD_Variable_Load_Example.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Rigid Dynamics** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the [Geometry Context tab](#). Navigate to the proper folder location and select `Mechanical_RBD_Variable_Load_Example.agdb`.



4. Select the **Automation** tab and select the **Scripting** option to open the **Mechanical Scripting** pane.

5. Select the **Open Script** option (from the [Editor \(p. 4\)](#) toolbar. Navigate to the proper folder location and select `Mechanical_RBD_Variable_Load_Example.py`.

6. Select the **Run Script** option (from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Section 1 - Set up the Tree Object Items

connections = Model.Connections
transient_rbd = Model.Analyses[0]
analysis_settings = transient_rbd.Children[0]
solution = transient_rbd.Solution

# Section 2 - Define the Named Selections

ns_torus_surface = DataModel.GetObjectsByName("NS_torus_surface")[0]
ns_cone_surface = DataModel.GetObjectsByName("NS_cone_surface")[0]

# Section 3 - Define the Torus Translational Joint

torus_translational_joint = connections.AddJoint()
torus_translational_joint.ConnectionType=JointScopingType.BodyToGround
torus_translational_joint.Type = JointType.Translational
torus_translational_joint.MobileLocation = ns_torus_surface

# Section 4 - Define the Cone Translational Joint

cone_translational_joint = connections.AddJoint()
cone_translational_joint.ConnectionType=JointScopingType.BodyToGround
```

```

cone_translational_joint.Type = JointType.Translational
cone_translational_joint.MobileLocation = ns_cone_surface

# Section 5 - Define the Spring connecting the Torus and the Cone

cone_torus_spring = connections.AddSpring()
cone_torus_spring.LongitudinalStiffness=Quantity("10 [N m-1]")
cone_torus_spring.Scope = SpringScopingType.BodyToBody
cone_torus_spring.ReferenceScopeLocation = ns_cone_surface
cone_torus_spring.MobileScopeLocation = ns_torus_surface

# Section 6 - Insert a Rigid Dynamics Measures Folder

rigid_dynamics_measures = ExtAPI.DataModel.CreateObject('MeasuresFolder', 'VariableLoad')

body_measures_attributes = ExtAPI.DataModel.GetUserObjects('VariableLoad')[1].Attributes["measurements"]
joint_measures_attributes = ExtAPI.DataModel.GetUserObjects('VariableLoad')[2].Attributes["measurements"]
derived_measures_attributes = ExtAPI.DataModel.GetUserObjects('VariableLoad')[3].Attributes["measurements"]

# Section 7 - Insert the Joint and the Derived Measures

joint_measures_attributes.append({'selection': torus_translational_joint.Name, 'variable': 'Translation', 'name': ''})
derived_measures_attributes.append({'selection': 'X', 'basemeasure': 'TorusTranslation', 'name': 'TorusTranslationX'})

# Section 8 - Insert a Measure Varying Joint Load to the Torus Translational Joint in X direction

measure_varying_joint_load_torus = transient_rbd.CreateLoadObject('MeasureJointLoad','VariableLoad')

measure_varying_joint_load_torus.Properties["JointSelection"].Value = str(torus_translational_joint.ObjectId)
measure_varying_joint_load_torus.Properties["JointDof"].Value = 'X'

# Section 9 - Define the Measure Output Type as a Table

measure_varying_joint_load_torus.Properties["OutputType"].Value = 'Table'
torus_table = measure_varying_joint_load_torus.Properties["OutputType"].Properties["Table"]
measure_value = torus_table.Properties["MeasureValue"]
magnitude = torus_table.Properties["Magnitude"]

torus_table.AddRow()
measure_value.Value = -0.1
magnitude.Value = 10
torus_table.SaveActiveRow()
torus_table.AddRow()
measure_value.Value = 0.1
magnitude.Value = -10
torus_table.SaveActiveRow()

# Section 10 - Define the Measures Selection

measure_selection = measure_varying_joint_load_torus.Properties["MeasurementSelections"]
measure_selection.UpdateStateFreq = UpdateStateFreqEnum.UpdateEachTime
measure = measure_selection.Properties["MeasurementSelection"]
measure_selection.AddRow()
measure.Value = 'TorusTranslationX'
measure_selection.SaveActiveRow()
measure_varying_joint_load_torus.NotifyChange()

# Section 11 - Insert Standard Earth Gravity

gravity = transient_rbd.AddEarthGravity()

# Section 12 - Insert Total Deformation result and Solve

total_deformation = solution.AddTotalDeformation()
solution.Solve(True)

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Steady State Electric Conduction Analysis

In this example, using the support files, you will insert an Steady State Electric Conduction analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will define and solve the analysis.

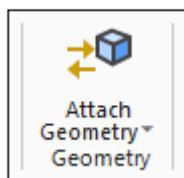
This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Steady_State_Electric_Conduction.agdb
- Steady_State_Electric_Conduction.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Static Structural** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the [Geometry Context tab](#). Navigate to the proper folder location and select Steady_State_Electric_Conduction.agdb.



4. Select the **Automation** tab and select the **Scripting** option to open the **Mechanical Scripting** pane.
5. Select the **Open Script** option () from the [Editor \(p. 4\)](#) toolbar. Navigate to the proper folder location and select Steady_State_Electric_Conduction.py.
6. Select the **Run Script** option () from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
#Section 1: Set up the Tree Object Items.  
#Section 2: Mesh.
```

```

#Section 3: Set up analysis settings.
#Section 4: Create named selections.
#Section 5: Set up voltage load.
#Section 6: Set up current load.
#Section 7: Set up thermal condition load.
#Section 8: Insert Results.
#Section 9: Solve the analysis.

#####
# Section 1 - Set up the Tree Object Items.
Geometry = Model.Geometry
Part1 = Geometry.Children[0]
Part2 = Geometry.Children[1]
Part3 = Geometry.Children[2]
Part4 = Geometry.Children[3]
ELEC = Model.Analyses[0]
ANA_SETTING = ELEC.Children[0]
SOLN = ELEC.Solution

#Section 2 - Mesh
MSH = Model.Mesh
MSH.ElementSize = Quantity("0.75[m]")

#Section 3 - Set up analysis settings
ELEC.EnvironmentTemperature = Quantity("0[C]")
ANA_SETTING.LineSearch = LineSearchType.On
ANA_SETTING.SolverType = SolverType.Direct
ANA_SETTING.NumberOfSteps = 2
ANA_SETTING.SetAutomaticTimeStepping(1, AutomaticTimeStepping.Off)
ANA_SETTING.SetDefineBy(1, TimeStepDefineByType.Substeps)
ANA_SETTING.SetNumberOfSubSteps(1, 10)

ANA_SETTING.SetAutomaticTimeStepping(2, AutomaticTimeStepping.Off)
ANA_SETTING.SetDefineBy(2, TimeStepDefineByType.Substeps)
ANA_SETTING.SetNumberOfSubSteps(2, 10)
ANA_SETTING.SetStoreResultsAt(2, TimePointsOptions.EquallySpacedPoints)
ANA_SETTING.SetStoreResultsAtValue(2, 5)

#Section 4 - Create named selection
Face1 = DataModel.Project.Model.AddNamedSelection()
Face1.ScopingMethod=GeometryDefineByType.Worksheet
Face1.Name = "Face1"
GEN_CRT1 = Face1.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
CRT1.Action=SelectionActionType.Add
CRT1.EntityType=SelectionType.GeoFace
CRT1.Criterion=SelectionCriterionType.LocationZ
CRT1.Operator=SelectionOperatorType.Equal
CRT1.Value=Quantity('20 [m]')
GEN_CRT1.Add(CRT1)
Face1.Activate()
Face1.Generate()

Face2 = DataModel.Project.Model.AddNamedSelection()
Face2.ScopingMethod=GeometryDefineByType.Worksheet
Face2.Name = "Face2"
GEN_CRT2 = Face2.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
CRT1.Action=SelectionActionType.Add
CRT1.EntityType=SelectionType.GeoFace
CRT1.Criterion=SelectionCriterionType.LocationZ
CRT1.Operator=SelectionOperatorType.Equal
CRT1.Value=Quantity('0 [m]')
GEN_CRT2.Add(CRT1)
Face2.Activate()
Face2.Generate()

Body1 = DataModel.Project.Model.AddNamedSelection()
Body1.ScopingMethod=GeometryDefineByType.Worksheet

```

```

Body1.Name = "Body1"
GEN_CRT3 = Body1.GenerationCriteria
CRT1 = Ansys.ACT.Automation.Mechanical.NamedSelectionCriterion()
CRT1.Active=True
CRT1.Action=SelectionActionType.Add
CRT1.EntityType=SelectionType.GeoBody
CRT1.Criterion=SelectionCriterionType.LocationZ
CRT1.Operator=SelectionOperatorType.LessThan
CRT1.Value=Quantity('22 [m]')
GEN_CRT3.Add(CRT1)
Body1.Activate()
Body1.Generate()

# Section 5 - Set up voltage load
VOLT_LD = ELEC.AddVoltage()
VOLT_LD.Location = Face1
VOLT_LD.Magnitude.Output.Formula = "100*time"
VOLT_LD.PhaseAngle = Quantity("45[deg]")
VOLT_LD.NumberOfSegments = 50

#Section 6 - Set up current load
CURR_LD = ELEC.AddCurrent()
CURR_LD.Location = Face2
CURR_LD.Magnitude.Inputs[0].DiscreteValues = [Quantity("0 [sec]"), Quantity("1 [sec]"), Quantity("2 [sec]")]
CURR_LD.Magnitude.Output.DiscreteValues = [Quantity("0[A]"), Quantity("100[A]"), Quantity("200[A]")]

#Section 7 - Set up thermal condition load
Thermal_Condition = ELEC.AddThermalCondition()
Thermal_Condition.Location = Body1
Thermal_Condition.Magnitude.Inputs[0].DiscreteValues = [Quantity("0 [sec]"), Quantity("1 [sec]"), Quantity("2 [sec]")]
Thermal_Condition.Magnitude.Output.DiscreteValues = [Quantity("0[C]"), Quantity("100[C]"), Quantity("200[C]")]

#Section 8 - Insert Results
ELEC_VOLT = SOLN.AddElectricVoltage()
ELEC_VOLT.DisplayTime = Quantity("1[s]")

ELEC_VOLT2 = SOLN.AddElectricVoltage()
ELEC_VOLT2.DisplayTime = Quantity("2[s]")

JOULE_HEAT = SOLN.AddJouleHeat()
JOULE_HEAT.DisplayTime = Quantity("1[s]")

JOULE_HEAT2 = SOLN.AddJouleHeat()
JOULE_HEAT2.DisplayTime = Quantity("2[s]")

DIR_ELEC_FLD_INT = SOLN.AddDirectionalElectricFieldIntensity()
DIR_ELEC_FLD_INT.NormalOrientation = NormalOrientationType.ZAxis
DIR_ELEC_FLD_INT.DisplayTime = Quantity("1[s]")

DIR_ELEC_FLD_INT2 = SOLN.AddDirectionalElectricFieldIntensity()
DIR_ELEC_FLD_INT2.NormalOrientation = NormalOrientationType.ZAxis
DIR_ELEC_FLD_INT2.DisplayTime = Quantity("2[s]")

DIR_CURR_DEN = SOLN.AddDirectionalCurrentDensity()
DIR_CURR_DEN.NormalOrientation = NormalOrientationType.ZAxis
DIR_CURR_DEN.DisplayTime = Quantity("1[s]")

DIR_CURR_DEN2 = SOLN.AddDirectionalCurrentDensity()
DIR_CURR_DEN2.NormalOrientation = NormalOrientationType.ZAxis
DIR_CURR_DEN2.DisplayTime = Quantity("2[s]")

REAC_PROBE = SOLN.AddEmagReactionProbe()
REAC_PROBE.BoundaryConditionSelection = VOLT_LD
REAC_PROBE.DisplayTime = Quantity("1[s]")

REAC_PROBE2 = SOLN.AddEmagReactionProbe()
REAC_PROBE2.BoundaryConditionSelection = VOLT_LD
REAC_PROBE2.DisplayTime = Quantity("2[s]")

#Section 9- Solve and extract results
SOLN.Solve(True)

```

```

ELEC_VOLT_VAL = ELEC_VOLT.Maximum.Value
ELEC_VOLT2_VAL = ELEC_VOLT2.Minimum.Value
JOULE_HEAT_VAL = JOULE_HEAT.Maximum.Value
JOULE_HEAT2_VAL = JOULE_HEAT2.Minimum.Value
DIR_ELEC_FLD_INT_VAL = DIR_ELEC_FLD_INT.Maximum.Value
DIR_ELEC_FLD_INT2_VAL = DIR_ELEC_FLD_INT2.Minimum.Value
DIR_CURR_DEN_VAL = DIR_CURR_DEN.Maximum.Value
DIR_CURR_DEN2_VAL = DIR_CURR_DEN2.Minimum.Value
REAC_PROBE_VAL = REAC_PROBE.Current.Value
REAC_PROBE2_VAL = REAC_PROBE2.Current.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Steady-State Thermal-Electric Conduction Analysis

In this example, using the support files, you will insert a Steady-State Thermal-Electric Conduction analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will define and solve the analysis.

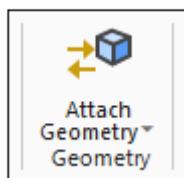
This example begins in the Mechanical application. It requires you to download the following Ansys DesignModeler and python files.

- Steady_State_Thermal_Electric_Conduction.agdb
- Steady_State_Thermal_Electric_Conduction.py

These files are available [here](#).

Procedure

1. Open Mechanical directly without importing a geometry or specifying an analysis type. This can be done through [Start Menu](#).
2. From the **Analysis** drop-down menu of the **Insert** group on the **Home** tab, insert a **Static Structural** system into the tree.
3. Select the **Geometry** object and select the **Attach Geometry** option from the **Geometry** group on the [Geometry Context Tab](#). Navigate to the proper folder location and select Steady_State_Thermal_Electric_Conduction.agdb.



4. Select the **Automation** tab and select the **Scripting** option to open the **Mechanical Scripting** pane.

5. Select the **Open Script** option () from the **Editor** (p. 4) toolbar. Navigate to the proper folder location and select Steady_State_Thermal_Electric_Conduction.py.

6. Select the **Run Script** option ( from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
#Section 1: Store the tree variables
#Section 2: Store the named selections
#section 3: Insert voltage load
#Section 4: Insert temperature and radiation loads
#Section 5: Insert results
#Section 6: Solve and review results
#Section 7: Define tabular Voltage & functional Voltage
#Section 8: Define tabular Temperature loads
#Section 9: Tabular Emissivity
#Section 10: Solve and review results
=====
#Section 1: Store the tree variables
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS
THERM_ELEC = Model.Analyses[0]
SOLN = THERM_ELEC.Solution

#Section 2: Store the named selections
NS_GRP = DataModel.Project.Model.NamedSelections
Face1 = NS_GRP.Children[0]
Face2 = NS_GRP.Children[1]
Face3 = NS_GRP.Children[2]
Face4 = NS_GRP.Children[3]
Face7 = NS_GRP.Children[6]

#section 3: Insert voltage load
VOLT_LD1 = THERM_ELEC.AddVoltage()
VOLT_LD1.Location = Face1
VOLT_LD2 = THERM_ELEC.AddVoltage()
VOLT_LD2.Location = Face3
VOLT_LD2.Magnitude.Output.DiscreteValues = [Quantity('0[V]'), Quantity('0.1[V]')]
VOLT_LD3 = THERM_ELEC.AddVoltage()
VOLT_LD3.Location = Face7

#Section 4: Insert temperature and radiation loads
TEMP = THERM_ELEC.AddTemperature()
TEMP.Location = Face2

RAD1 = THERM_ELEC.AddRadiation()
RAD1.Location = Face4
RAD1.Correlation = RadiationType.SurfaceToSurface

RAD2 = THERM_ELEC.AddRadiation()
RAD2.Location = Face7
RAD2.Correlation = RadiationType.SurfaceToSurface

#Section 5: Insert results
TEMP_RST1 = SOLN.AddTemperature()
TEMP_RST2 = SOLN.AddTemperature()
TEMP_RST2.Location = Face4
TEMP_RST3 = SOLN.AddTemperature()
TEMP_RST3.Location = Face7

RAD_PROBE1 = SOLN.AddRadiationProbe()
RAD_PROBE1.BoundaryConditionSelection = RAD1
RAD_PROBE2 = SOLN.AddRadiationProbe()
RAD_PROBE2.BoundaryConditionSelection = RAD2

#Section 6: Solve and review results
SOLN.Solve(True)
TEMP_RST1_VAL = TEMP_RST1.Maximum.Value
```

```

TEMP_RST2_VAL = TEMP_RST2.Maximum.Value
TEMP_RST3_VAL = TEMP_RST3.Maximum.Value
RAD_PROBE1_VAL = RAD_PROBE1.EmittedRadiation.Value
RAD_PROBE2_VAL = RAD_PROBE2.EmittedRadiation.Value

#Section 7: Define tabular Voltage & functional Voltage
VOLT_LD2.Magnitude.Output.DiscreteValues=[Quantity('0.05[V]'), Quantity('0.11[V]')]
VOLT_LD3.Magnitude.Output.Formula = '0.1*time'
VOLT_LD3.PhaseAngle = Quantity("45[deg]")
VOLT_LD3.NumberOfSegments = 50

#Section 8: Define tabular Temperature loads
TEMP.Magnitude.Output.DiscreteValues=[Quantity('22[C]'), Quantity('60[C]')]

#Section 9: Tabular Emissivity
RAD1.Emissivity.Output.DiscreteValues=[Quantity('0.4'), Quantity('0.7')]

#Section 10: Solve and review results
SOLN.Solve(True)
TEMP_RST1_VAL = TEMP_RST1.Maximum.Value
TEMP_RST2_VAL = TEMP_RST2.Maximum.Value
TEMP_RST3_VAL = TEMP_RST3.Maximum.Value
RAD_PROBE1_VAL = RAD_PROBE1.EmittedRadiation.Value
RAD_PROBE2_VAL = RAD_PROBE2.EmittedRadiation.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Magnetostatic Analysis

In this example, using the support files, you will insert a Magnetostatic analysis object into an undefined Mechanical session and execute a sequence of python journal commands that will define and solve the analysis.

This example begins in the Mechanical application. It requires you to download the following files.

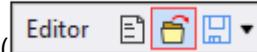
- Magnetostatic_001.agdb
- Magnetostatic_001.py
- Magnetostatic_001_Mat1.xml
- Magnetostatic_001_Mat2.xml

These files are available [here](#).

Procedure

1. Open Workbench and insert a **Magnetostatic** system into the Project Schematic.
2. Double-click the **Engineering Data** cell to open the workspace.
3. Select **File > Import Engineering Data**, navigate to the proper folder location and select Magnetostatic_001_Mat1.xml. Repeat this action for the Magnetostatic_001_Mat2.xml file.
4. Return to the **Project** tab.

5. Right-click the **Geometry** cell and select **Import Geometry > Browse** and then navigate to the proper folder location and select Magnetostatic_001.agdb.
6. Open Mechanical: right-click the **Model** cell and select **Edit**.
7. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.



8. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select Magnetostatic_001.py.



9. Select the **Run Script** option () from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Scenario1 - Set up Unit System: Set up the Unit Systems to Metric (m, kg, N, s, V, A),
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

# Scenario 2 - Set up the Tree Object Items: Define the parts, connections, mesh, Magnetostatic system, the analysis
GEOMETRY = Model.Geometry
BODY1 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'air_body'][0]
BODY2 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'conductor'][0]
MESH = Model.Mesh
MAGNETOSTATIC = DataModel.AnalysisByName("Magnetostatic")
ANSLYSIS_SETTINGS = MAGNETOSTATIC.AnalysisSettings
SOLUTION = MAGNETOSTATIC.Solution

# Scenario 3 - Define Named Selections: Define the Named selections that will be used to define the Mesh Controls,
NS_VOLUME1 = DataModel.GetObjectsByName("NS_Volume1")[0]
NS_VOLUME2 = DataModel.GetObjectsByName("NS_Volume2")[0]
NS_VOLUME = DataModel.GetObjectsByName("NS_Volume")[0]
NS_FACES5 = DataModel.GetObjectsByName("Faces5")[0]
NS_FACE1 = DataModel.GetObjectsByName("Face1")[0]
NS_FACE2 = DataModel.GetObjectsByName("Face2")[0]

# Scenario 4 Create Coordinate System: Create Coordinate System that will be used to define the results.
COORDINATE_SYSTEMS = Model.CoordinateSystems
GCS = COORDINATE_SYSTEMS.Children[0]
LCS = COORDINATE_SYSTEMS.AddCoordinateSystem()
LCS.OriginX = Quantity('0 [m]')
LCS.PrimaryAxis = CoordinateSystemAxisType.PositiveYAxis
LCS.AddTransformation(TransformationType.Rotation,CoordinateSystemAxisType.PositiveZAxis)
LCS.SetTransformationValue(1,-120)

# Scenario 5 Assign materials: Assign materials to bodies.

BODY1.Material = 'Emag_Air'
BODY2.Material = 'Emag_Copper_Alloy'

# Scenario 6 - Define Mesh Settings: Assign Body Sizing with element size of 0.5 m.

BODY_SIZING = MESH.AddSizing()
BODY_SIZING.Location = NS_VOLUME
BODY_SIZING.ElementSize = Quantity('0.5 [m]')
MESH.GenerateMesh()

# Scenario 7 - Insert Magnetostatic loads: Insert and Magnetic Flux Parallel, Source Conductor with conductor type a

MAGNETIC_FLUX_PARALLEL = MAGNETOSTATIC.AddMagneticFluxParallel()
```

```

MAGNETIC_FLUX_PARALLEL.Location = NS_FACES5

SOURCE_CONDUCTOR = MAGNETOSTATIC.AddSourceConductor()
SOURCE_CONDUCTOR.Location = NS_VOLUME2
SOURCE_CONDUCTOR.ConductorType = SourceConductorType.Solid
SOURCE_CONDUCTOR.NumberOfTurns = 1

SOURCE_CONDUCTOR_CURRENT = SOURCE_CONDUCTOR.AddCurrent()
SOURCE_CONDUCTOR_CURRENT.Location = NS_FACE1
SOURCE_CONDUCTOR_CURRENT.Magnitude.Output.DiscreteValues=[Quantity('0 [A]'), Quantity('-120 [A]')]

SOURCE_CONDUCTOR_VOLTAGE = SOURCE_CONDUCTOR.AddVoltage()
SOURCE_CONDUCTOR_VOLTAGE.Location = NS_FACE2

# Scenario 8 - Insert Results: Insert Directional Magnetic Field Intensity, Directional Magnetic Flux Density, Magn

DIRECTIONAL_MAGNETIC_FIELD_INTENSITY = SOLUTION.AddDirectionalMagneticFieldIntensity()
DIRECTIONAL_MAGNETIC_FIELD_INTENSITY.Location = NS_VOLUME1
DIRECTIONAL_MAGNETIC_FIELD_INTENSITY.NormalOrientation = NormalOrientationType.YAxis

DIRECTIONAL_MAGNETIC_FLUX_DENSITY = SOLUTION.AddDirectionalMagneticFluxDensity()
DIRECTIONAL_MAGNETIC_FLUX_DENSITY.Location = NS_VOLUME1
DIRECTIONAL_MAGNETIC_FLUX_DENSITY.NormalOrientation = NormalOrientationType.YAxis

DIRECTIONAL_FORCE01 = SOLUTION.AddMagneticDirectionalForces()
DIRECTIONAL_FORCE01.Location = NS_VOLUME2

DIRECTIONAL_FORCE02 = SOLUTION.AddMagneticDirectionalForces()
DIRECTIONAL_FORCE02.Location = NS_VOLUME2
DIRECTIONAL_FORCE02.NormalOrientation = NormalOrientationType.YAxis

DIRECTIONAL_FORCE03 = SOLUTION.AddMagneticDirectionalForces()
DIRECTIONAL_FORCE03.Location = NS_VOLUME2
DIRECTIONAL_FORCE03.NormalOrientation = NormalOrientationType.ZAxis

DIRECTIONAL_FORCE04 = SOLUTION.AddMagneticDirectionalForces()
DIRECTIONAL_FORCE04.Location = NS_VOLUME2
DIRECTIONAL_FORCE04.NormalOrientation = NormalOrientationType.YAxis
DIRECTIONAL_FORCE04.CoordinateSystem = LCS

DIRECTIONAL_FORCE05 = SOLUTION.AddMagneticDirectionalForces()
DIRECTIONAL_FORCE05.Location = NS_VOLUME2
DIRECTIONAL_FORCE05.NormalOrientation = NormalOrientationType.ZAxis
DIRECTIONAL_FORCE05.CoordinateSystem = LCS

FORCE_SUMMATION01 = SOLUTION.AddForceSummationProbe()
FORCE_SUMMATION01.GeometryLocation = NS_VOLUME2

FORCE_SUMMATION02 = SOLUTION.AddForceSummationProbe()
FORCE_SUMMATION02.GeometryLocation = NS_VOLUME2
FORCE_SUMMATION02.ResultSelection = ProbeDisplayFilter.YAxis

FORCE_SUMMATION03 = SOLUTION.AddForceSummationProbe()
FORCE_SUMMATION03.GeometryLocation = NS_VOLUME2
FORCE_SUMMATION03.ResultSelection = ProbeDisplayFilter.ZAxis

FORCE_SUMMATION04 = SOLUTION.AddForceSummationProbe()
FORCE_SUMMATION04.GeometryLocation = NS_VOLUME2
FORCE_SUMMATION04.ResultSelection = ProbeDisplayFilter.YAxis
FORCE_SUMMATION04.Orientation = LCS

FORCE_SUMMATION05 = SOLUTION.AddForceSummationProbe()
FORCE_SUMMATION05.GeometryLocation = NS_VOLUME2
FORCE_SUMMATION05.ResultSelection = ProbeDisplayFilter.ZAxis
FORCE_SUMMATION05.Orientation = LCS

TORQUE01 = SOLUTION.AddTorqueProbe()
TORQUE01.GeometryLocation = NS_VOLUME2

TORQUE02 = SOLUTION.AddTorqueProbe()
TORQUE02.GeometryLocation = NS_VOLUME2

```

```

TORQUE02.ResultSelection = ProbeDisplayFilter.YAxis

TORQUE03 = SOLUTION.AddTorqueProbe()
TORQUE03.GeometryLocation = NS_VOLUME2
TORQUE03.ResultSelection = ProbeDisplayFilter.ZAxis

TORQUE04 = SOLUTION.AddTorqueProbe()
TORQUE04.GeometryLocation = NS_VOLUME2
TORQUE04.ResultSelection = ProbeDisplayFilter.YAxis
TORQUE04.Orientation = LCS

TORQUE05 = SOLUTION.AddTorqueProbe()
TORQUE05.GeometryLocation = NS_VOLUME2
TORQUE05.ResultSelection = ProbeDisplayFilter.ZAxis
TORQUE05.Orientation = LCS

# Scenario 9 - Solve and Define the Results: Solve the system and set the results variables. Note the Scripting window.

SOLUTION.Solve(True)

DIRECTIONAL_MAGNETIC_FIELD_INTENSITY_MAX = DIRECTIONAL_MAGNETIC_FIELD_INTENSITY.Maximum.Value
DIRECTIONAL_MAGNETIC_FLUX_DENSITY_MAX = DIRECTIONAL_MAGNETIC_FLUX_DENSITY.Maximum.Value

FORCE_SUMMATION01_X_AXIS = FORCE_SUMMATION01.XAxis.Value
FORCE_SUMMATION02_Y_AXIS = FORCE_SUMMATION02.YAxis.Value
FORCE_SUMMATION03_Z_AXIS = FORCE_SUMMATION03.ZAxis.Value
FORCE_SUMMATION04_Y_AXIS = FORCE_SUMMATION04.YAxis.Value
FORCE_SUMMATION05_Z_AXIS = FORCE_SUMMATION05.ZAxis.Value

TORQUE01_X_AXIS = TORQUE01.XAxis.Value
TORQUE02_Y_AXIS = TORQUE02.YAxis.Value
TORQUE03_Z_AXIS = TORQUE03.ZAxis.Value
TORQUE04_Y_AXIS = TORQUE04.YAxis.Value
TORQUE05_Z_AXIS = TORQUE05.ZAxis.Value

```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Post Processing Options

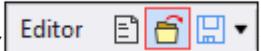
Using a solved project, this example examines APIs for result post processing and graphical manipulation actions. This example requires you to download the following project and python files.

- Mechanical_Post_Example.wbpz
- Mechanical_Post_Example.py

These files are available [here](#).

Procedure

1. Open the project file Mechanical_Post_Example.wbpz in the Workbench application.
2. Right-click the **Model** cell in the Project Schematic and the Edit option. This opens the project in Mechanical.
3. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.

4. Select the **Open Script** option () from the [Editor \(p. 4\)](#) toolbar. Navigate to the proper folder location and select Mechanical_Post_Graphics_Example_001.py.

5. Select the **Run Script** option () from the [Editor \(p. 4\)](#) toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# -*- coding: UTF-8 -*-
#Scenario 1: Set up the Tree objects items and API base definitions. This step is necessary for all the API functions
GEOM=Model.Geometry
MESH=Model.Mesh
STATIC_STRUCTURAL=Model.Analyses[0]
SOLUTION=STATIC_STRUCTURAL.Solution
CAMERA=Graphics.Camera
GRAPHICS_SETTINGS= Ansys.Mechanical.Graphics.GraphicsImageExportSettings()
LEGEND_SETTINGS=Ansys.Mechanical.Graphics.Tools.CurrentLegendSettings()
GLOBAL_SETTINGS=Graphics.GlobalLegendSettings
VIEW_OPTIONS=Graphics.ViewOptions
FOLDER_PATH='E:\\temp'

#Scenario 2: Insert results. This step would insert the results that we would use to demonstrate the different API functions
TOTAL_DEFORMATION_RESULT=SOLUTION.AddTotalDeformation()
#Scenario 3: Setup the units system.
ExtAPI.Application.ActiveUnitSystem=MechanicalUnitSystem.StandardMKS
#Scenario 4: Clear and Solve
SOLUTION.ClearGeneratedData()
SOLUTION.Solve()
#Scenario 5: Change the Camera angles, Orientation and Export images. Change the export settings and repeat for a few more
#To fit the model to screen.
TOTAL_DEFORMATION_RESULT.Activate()
CAMERA.SetFit()
#To change the view to Front, back, Top etc., and export images with default settings. The images can be saved at a specific location
CAMERA.SetSpecificViewOrientation(ViewOrientationType.Front)
Graphics.ExportImage(FOLDER_PATH+'\\DEF1.png', GraphicsImageExportFormat.PNG, GRAPHICS_SETTINGS)

CAMERA.SetSpecificViewOrientation(ViewOrientationType.Back)
Graphics.ExportImage(FOLDER_PATH+'\\DEF2.png', GraphicsImageExportFormat.PNG, GRAPHICS_SETTINGS)

CAMERA.SetSpecificViewOrientation(ViewOrientationType.Right)
Graphics.ExportImage(FOLDER_PATH+'\\DEF3.png', GraphicsImageExportFormat.PNG, GRAPHICS_SETTINGS)

CAMERA.SetSpecificViewOrientation(ViewOrientationType.Left)
Graphics.ExportImage(FOLDER_PATH+'\\DEF4.png', GraphicsImageExportFormat.PNG, GRAPHICS_SETTINGS)

CAMERA.SetSpecificViewOrientation(ViewOrientationType.Top)
Graphics.ExportImage(FOLDER_PATH+'\\DEF5.png', GraphicsImageExportFormat.PNG, GRAPHICS_SETTINGS)

CAMERA.SetSpecificViewOrientation(ViewOrientationType.Bottom)
Graphics.ExportImage(FOLDER_PATH+'\\DEF6.png', GraphicsImageExportFormat.PNG, GRAPHICS_SETTINGS)

CAMERA.SetSpecificViewOrientation(ViewOrientationType.Iso)
Graphics.ExportImage(FOLDER_PATH+'\\DEF7.png', GraphicsImageExportFormat.PNG, GRAPHICS_SETTINGS)

#Orient the model to non-standard orientation

#Change the focal point (Works like Pan)
CAMERA.FocalPoint = Point([0.050000, 0.150000, 0.180000], 'm')

#Change the camera angle (Works like Rotate)
CAMERA.ViewVector = Vector3D(0.75331, 0.58096, -0.308235)
CAMERA.UpVector = Vector3D(-0.523344, 0.813377, 0.254025)
```

```
#Change the height and width of Scene (Works like Zoom)
CAMERA.SceneHeight = Quantity(0.477099, 'm')
CAMERA.SceneWidth = Quantity(0.358609, 'm')

#Define the Image export settings. First define Image resolutions
GRAPHICS_SETTINGS.Width = 1920
GRAPHICS_SETTINGS.Height = 1080

#Define whether image would contain legend or Image only
GRAPHICS_SETTINGS.Capture=GraphicsCaptureType.ImageOnly

#Remove the defaults and set the background to White
GRAPHICS_SETTINGS.CurrentGraphicsDisplay = False
GRAPHICS_SETTINGS.Background=GraphicsBackgroundType.White

Graphics.ExportImage(FOLDER_PATH+'\\DEF8.png', GraphicsImageExportFormat.PNG, GRAPHICS_SETTINGS)

#Orient the legend horizontally, hide ruler, date and Time and triads.
GLOBAL_SETTINGS.LegendOrientation=LegendOrientationType.Horizontal
GLOBAL_SETTINGS.ShowDateAndTime=False
VIEW_OPTIONS.ShowTriad=False
VIEW_OPTIONS.ShowRuler=False

#Scenario 6: Change the Legend to display with different bands, scale, Color Scheme etc., and export those images as
#Change the number of bands in the legend to 4, color scale to Greyscale and change upper bound value of band 1 to a
LEGEND_SETTINGS.NumberOfBands=4
LEGEND_SETTINGS.ColorScheme=LegendColorSchemeType.GrayScale
LEGEND_SETTINGS.SetUpperBound(1,Quantity(0.4,'mm'))

#Scenario 7: Create section plane
CAMERA.SetSpecificViewOrientation(ViewOrientationType.Right)
section_plane = Ansys.Mechanical.Graphics.SectionPlane(Point([0.050000, 0.160203, 0.180000], Graphics.Unit), Vector[0,0,1])
Graphics.SectionPlanes.Add(section_plane)

#Scenario 8: Export results as Text file/csv to User defined location.
TOTAL_DEFORMATION_RESULT.Activate()
TOTAL_DEFORMATION_RESULT.ExportToTextFile(FOLDER_PATH+'\\Txt1.txt')

#Scenario 9: Resetting to defaults. Execute this to unmake all the changes made in previous scenarios
LEGEND_SETTINGS.Reset()
Graphics.SectionPlanes.RemoveAt(0)
GLOBAL_SETTINGS.ShowDateAndTime=True
VIEW_OPTIONS.ShowTriad=True
VIEW_OPTIONS.ShowRuler=True
```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Post Processing User Defined Results

Using a solved project, this example examines APIs for post processing User Defined Results and graphical manipulation actions. This example requires you to download the following project and python files.

- Post_Processing_User_Defined_Results.wbpz
- Post_Processing_User_Defined_Results.py

These files are available [here](#).

Procedure

1. Open the project file Post_Processing_User_Defined_Results.wbpz in the Workbench application.
2. Right-click the **Model** cell in the **Project Schematic** and the **Edit** option. This opens the project in Mechanical.
3. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.



4. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select Post_Processing_User_Defined_Results.py.



5. Select the **Run Script** option () from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# -*- coding: UTF-8 -*-
#Scenario 1: Set up the Tree objects items and API base definitions. This step is necessary for all the API functions
GEOM=Model.Geometry
MESH=Model.Mesh
STATIC_STRUCTURAL=Model.Analyses[0]
SOLUTION=STATIC_STRUCTURAL.Solution

#Scenario 2: Add different results from Tree. In this example we will add Total Deformation result, von Mises stress result and Normal Stress result.
TOTAL_DEFORMATION_RESULT=SOLUTION.AddTotalDeformation()
EQUIVALENT_STRESS_RESULT=SOLUTION.AddEquivalentStress()
NORMAL_STRESS=SOLUTION.AddNormalStress()

#Scenario 3: Modify display options for inserted results.
#Modify the result display option from Averaged(default) to Unaveraged.
EQUIVALENT_STRESS_RESULT.DisplayOption=ResultAveragingType.Unaveraged
#Modify the Orientation to Y-Axis for Normal stress results (only applicable with certain results)
NORMAL_STRESS.NormalOrientation=NormalOrientationType.YAxis
#Modify the result display time to 0.4 s.
TOTAL_DEFORMATION_RESULT.DisplayTime=Quantity(0.4,'s')
#Get Result values, maximum of Total Deformation Results
VAL1=TOTAL_DEFORMATION_RESULT.Maximum.Value

#Scenario 4: Add User Defined results. In this example we will add USUM, SEQV and SX to reflect the regular results
USUM=SOLUTION.AddUserDefinedResult()
USUM.Expression='USUM'

SEQV= SOLUTION.AddUserDefinedResult()
SEQV.Expression='SEQV'

SX= SOLUTION.AddUserDefinedResult()
SX.Expression='SX'

#Scenario 5: Modify display options for user defined results
#Modify the result output unit category.
USUM.OutputUnit=UnitCategoryType.Length
SEQV.OutputUnit=UnitCategoryType.Stress
SX.OutputUnit=UnitCategoryType.Stress

#Modify the result Averaging option across bodies
SEQV.AverageAcrossBodies=True

#Modify the Identifier
```

```
SX.Identifier='SX_ID'

#Modify the result display by option
USUM.By=SetDriverStyle.ResultSet

#Scenario 6: Duplicate, Clear, Evaluate results. Create Results at All sets
USUM2=USUM.DuplicateWithoutResults()
SEQV.ClearGeneratedData()
SX.CreateResultsAtAllSets()
USUM.RenameBasedOnDefinition()
SEQV.RenameBasedOnDefinition()
SX.RenameBasedOnDefinition()
SOLUTION.EvaluateAllResults()

#Scenario 7: Use PlotData API to extract useful information from results. Here we show a snippet to access values at
SOLUTION.EvaluateAllResults()
PLOT_DATA_RESULT= EQUIVALENT_STRESS_RESULT.PlotData

def findNodeResult(nodeID,plotDataResult):
    nodes=plotDataResult ['Node']
    resultValue=plotDataResult ['Values']
    if nodeID in nodes:
        for index in range(len(nodes)):
            if nodes[index]==nodeID:
                return resultValue[index]
        else:
            print("Given NodeID doesn't exist in result set")
    #Find the result value associated for node number 3 using
    VALUE2=findNodeResult(3, PLOT_DATA_RESULT)
    print('The result value at node 3 is '+str(VALUE2))

#Scenario 8: Add figures under the results.
FIGURE=EQUIVALENT_STRESS_RESULT.AddFigure()
```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Random Vibration Analysis

In this example, using the support files, you will open a Random Vibration analysis in Mechanical and execute a series of scripts to solve and complete an entire analysis. First, you must prepare certain aspects of the analysis on the Workbench Project Schematic. This example begins in the Workbench application. It requires you to download the following Ansys DesignModeler and python files.

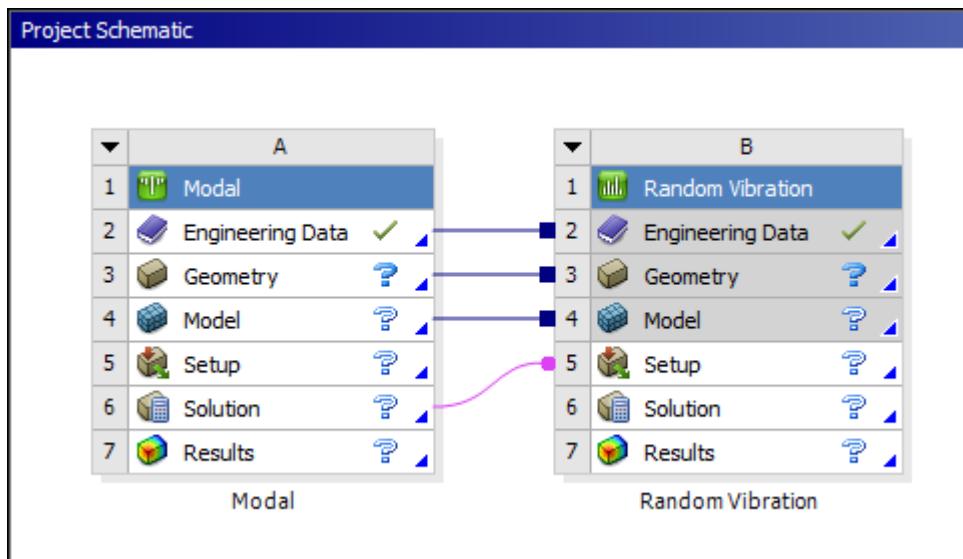
- Mechanical_Random_Vibration_Example012_Geometry.agdb
- Mechanical_Random_Vibration_Example012_Script.py

These files are available [here](#).

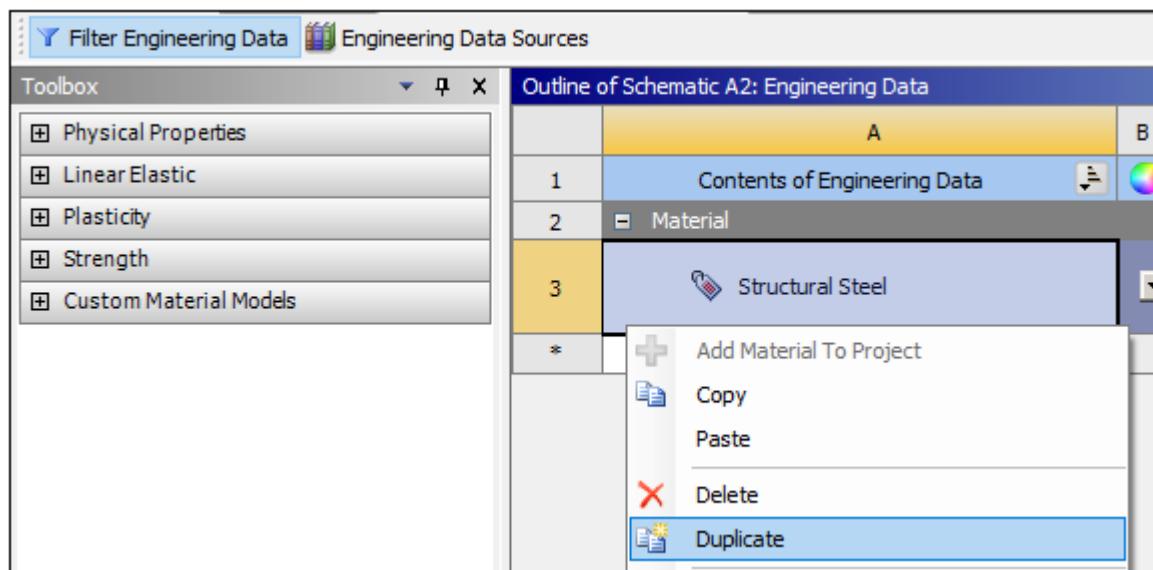
Procedure

The following procedure assumes you understand how to add a system in Workbench and make changes to system cells and properties as well as saving the support files to a known location.

1. Add a **Modal** analysis to the **Project Schematic**.
2. Right-click the **Solution** cell of the **Modal** analysis and select **Transfer Data To New > Random Vibration**. This links the two necessary systems.



3. This analysis requires a new material. Double-click the **Engineering Data** cell of the **Modal** analysis to open the Engineering Data workspace.
4. Right-click **Structural Steel** material and select **Duplicate**.



5. Double-click the new field and rename the material "Rod_Mat."

The screenshot shows the ANSYS Project Schematic interface. On the left is a 'Toolbox' containing categories like Field Variables, Physical Properties, Linear Elastic, Plasticity, Strength, and Custom Material Models. The main area displays the 'Outline of Schematic A2: Engineering Data' table:

	A	B	C	D	E
1	Contents of Engineering Data			Description	
2	Material				
3	Structural Steel				G Fatigue Data at zero mean stress comes from 1998 ASME BPV Code, Section 8, Div 2, Table 5-110.1
4	Rod_Mat				G Fatigue Data at zero mean stress comes from 1998 ASME BPV Code, Section 8, Div 2, Table 5-110.1
*	Click here to add a new material				

Below this is the 'Properties of Outline Row 4: Rod_Mat' dialog box:

	A	B	C	D	E
1	Property	Value	Unit		
2	Material Field Variables				
3	Density	7850	kg m ⁻³		
4	Isotropic Secant Coefficient of Thermal Expansion				
6	Isotropic Elasticity				
7	Derive from	Young's Mod...			
8	Young's Modulus	2E+11	Pa		
9	Poisson's Ratio	0.3			
10	Bulk Modulus	1.6667E+11	Pa		
11	Shear Modulus	7.6923E+10	Pa		

6. Set the Density to 0.1 Kg/m³.
7. Open the Isotropic Elasticity property and set the following values:
 - Young's Modulus = 2E+8 Pa
 - Poisson's Ratio = 0
8. Return to the **Project Schematic** and select the **Geometry** cell of the **Modal** system. Right-click and select **Properties**.
9. Under the **Basic Geometry Options** group of the properties, activate the **Named Selections** property.

The screenshot shows the ANSYS Project Schematic interface. The 'Project Schematic' window on the left shows a hierarchical structure with nodes like Modal, Engineering Data, Geometry, Model, Setup, Solution, and Results. The 'Properties of Schematic A3: Geometry' dialog box on the right displays the following table:

	A	B
1	Property	Value
2	General	
6	Notes	
8	Used Licenses	
10	Basic Geometry Options	
11	Solid Bodies	<input checked="" type="checkbox"/>
12	Surface Bodies	<input checked="" type="checkbox"/>
13	Line Bodies	<input type="checkbox"/>
14	Parameters	Independent
15	Parameter Key	ANS;DS
16	Attributes	<input type="checkbox"/>
17	Named Selections	<input checked="" type="checkbox"/>
18	Named Selection Key	NS
19	Material Properties	<input type="checkbox"/>

10. Right-click the **Geometry** cell and select **Import Geometry > Browse** and select the `Mechanic-a1_Random_Vibration_Example012_Geometry.agdb` file.
11. Right-click the **Model** cell and select **Edit**. This action launches the Mechanical application.
12. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.

13. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select **Mechanical_Random_Vibration_Example012_Script.py**.

14. Select the **Run Script** option () from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Scenario 1 Store main Tree Object items
MODEL = Model
GEOMETRY = Model.Geometry
ROD1 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Rod1'][0]
BLOCK1 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Block1'][0]
ROD2 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Rod2'][0]
BLOCK2 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Block2'][0]
ROD3 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Rod3'][0]
BLOCK3 = [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Block3'][0]

MATERIALS = MODEL.Materials
MATERIAL_RM = [i for i in MATERIALS.GetChildren[Ansys.ACT.Automation.Mechanical.Material](True) if i.Name == 'Rod_M'
MATERIAL_SS = [i for i in MATERIALS.GetChildren[Ansys.ACT.Automation.Mechanical.Material](True) if i.Name == 'Struct'

COORDINATE_SYSTEMS = Model.CoordinateSystems
GLOBAL_COORDINATE_SYSTEM = [i for i in COORDINATE_SYSTEMS.GetChildren[Ansys.ACT.Automation.Mechanical.CoordinateSys

MESH = Model.Mesh

NAMED_SELECTIONS = ExtAPI.DataModel.Project.Model.NamedSelections
NS_BLOCK_SIDES = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.
NS_ROD_SIDES = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.
NS_BASE = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.
NS_BLOCK_ROD_SIDES = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.
NS_BLOCK1_SIDES = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.
NS_BLOCK2_SIDES = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.
NS_BLOCK3_SIDES = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.
NS_BLOCK1_VERTEX = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.
NS_BLOCK2_VERTEX = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.
NS_BLOCK3_VERTEX = [i for i in NAMED_SELECTIONS.GetChildren[Ansys.ACT.Automation.Mechanical.NamedSelection](True) if i.

MODAL = DataModel.AnalysisByName("Modal")
ANALYSIS_SETTINGS_MODAL = MODAL.AnalysisSettings
SOLUTION_MODAL = MODAL.Solution

PSD = DataModel.AnalysisByName("Random Vibration")
ANALYSIS_SETTINGS_PSD = PSD.AnalysisSettings
SOLUTION_PSD = PSD.Solution

# Scenario 2 Set Display Unit
ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

# Scenario 3 Set Brick Integration Scheme for all bodies
GEOMETRY.Activate()
GEOMETRY.ElementControl = ElementControl.Manual

ROD1Activate()
ROD1.BrickIntegrationScheme = BrickIntegrationScheme.Full

BLOCK1Activate()
BLOCK1.BrickIntegrationScheme = BrickIntegrationScheme.Full

ROD2Activate()
ROD2.BrickIntegrationScheme = BrickIntegrationScheme.Full
```

```

BLOCK2.Activate()
BLOCK2.BrickIntegrationScheme = BrickIntegrationScheme.Full

ROD3.Activate()
ROD3.BrickIntegrationScheme = BrickIntegrationScheme.Full

BLOCK3.Activate()
BLOCK3.BrickIntegrationScheme = BrickIntegrationScheme.Full

# Scenario 4 Assign material to Rod bodies
ROD1.Activate()
ROD1.Material = MATERIAL_RM.Name

ROD2.Activate()
ROD2.Material = MATERIAL_RM.Name

ROD3.Activate()
ROD3.Material = MATERIAL_RM.Name

# Scenario 5 Add local Coordinate System and Remote Point
COORDINATE_SYSTEMSActivate()
COORDINATE_SYSTEM = COORDINATE_SYSTEMS.AddCoordinateSystem()
COORDINATE_SYSTEM.OriginLocation = NS_BLOCK1_SIDES

MODEL.Activate()
REMOTE_POINT = MODEL.AddRemotePoint()
REMOTE_POINT.Location = NS_BLOCK2_SIDES
REMOTE_POINT.Behavior=LoadBehavior.Deformable

# Scenario 6 Define global mesh size and generate mesh
MESH.Activate()
MESH.ElementSize = Quantity('0.105 [m]')
MESH.GenerateMesh()

# Scenario 7 Define Analysis Settings for Modal system
# Modify Analysis Settings of Modal to find first three modes only
ANALYSIS_SETTINGS_MODAL.Activate()
ANALYSIS_SETTINGS_MODAL.MaximumModesToFind = 3

# Scenario 8 Define boundary conditions for Modal system
MODAL.Activate()
FIXED_SUPPORT = MODAL.AddFixedSupport()
FIXED_SUPPORT.Location = NS_BASE

MODAL.Activate()
FRICTIONLESS_SUPPORT = MODAL.AddFrictionlessSupport()
FRICTIONLESS_SUPPORT.Location = NS_BLOCK_ROD_SIDES

# Scenario 9 Add results for Modal system
# Add Total Deformation results in Modal analysis for each modes
SOLUTION_MODAL.Activate()
TOTAL_DEFORMATION_MODAL = SOLUTION_MODAL.AddTotalDeformation()
TOTAL_DEFORMATION_MODAL.Mode = 1

TOTAL_DEFORMATION2_MODAL = SOLUTION_MODAL.AddTotalDeformation()
TOTAL_DEFORMATION2_MODAL.Mode = 2

TOTAL_DEFORMATION2_MODAL = SOLUTION_MODAL.AddTotalDeformation()
TOTAL_DEFORMATION2_MODAL.Mode = 3

# Scenario 10 Define Analysis Settings for Random Vibration system
# Modify Analysis Settings of Random Vibration to enable Velocity Acceleration calculations and add Damping
ANALYSIS_SETTINGS_PSD.Activate()
ANALYSIS_SETTINGS_PSD.CalculateVelocity = True
ANALYSIS_SETTINGS_PSD.CalculateAcceleration = True

ANALYSIS_SETTINGS_PSD.ConstantDamping=ConstantDampingType.Manual
ANALYSIS_SETTINGS_PSD.ConstantDampingRatio = 0
ANALYSIS_SETTINGS_PSD.StiffnessCoefficient = 0.0028

```

```

# Scenario 11 Add PSD Acceleration for Random Vibration system
PSD.Activate()
PSD_ACCELERATION = PSD.AddPSDAcceleration()
PSD_ACCELERATION.BoundaryCondition = PSDBoundaryConditionSelectionType.AllFixedSupports
PSD_ACCELERATION.LoadData.Inputs[0].DiscreteValues = [Quantity("1.e-003 [Hz]"), Quantity("1000 [Hz]")]
PSD_ACCELERATION.LoadData.Output.DiscreteValues = [Quantity("0.62832 [(m sec^-2)^2 Hz^-1]"), Quantity("0.62832 [(m sec^-2)^2 Hz^-1]")]
PSD_ACCELERATION.Direction = NormalOrientationType.XAxis

# Scenario 12 Add regular and probe results for PSD analysis
SOLUTION_PSDActivate()
DIRECTIONAL_DEFORMATION_PSD = SOLUTION_PSD.AddDirectionalDeformation()
DIRECTIONAL_DEFORMATION_PSD.NormalOrientation=NormalOrientationType.XAxis
DIRECTIONAL_DEFORMATION_PSD.ScaleFactor = ScaleFactorType.Sigma1

DIRECTIONAL_VELOCITY_PSD = SOLUTION_PSD.AddDirectionalVelocityPSD()
DIRECTIONAL_VELOCITY_PSD.NormalOrientation=NormalOrientationType.XAxis
DIRECTIONAL_VELOCITY_PSD.ScaleFactor = ScaleFactorType.Sigma2

DIRECTIONAL_ACCELERATION_PSD = SOLUTION_PSD.AddDirectionalAccelerationPSD()
DIRECTIONAL_ACCELERATION_PSD.NormalOrientation=NormalOrientationType.XAxis
DIRECTIONAL_ACCELERATION_PSD.ScaleFactor = ScaleFactorType.Sigma3

NORMAL_ELASTIC_STRAIN_PSD = SOLUTION_PSD.AddNormalElasticStrain()
NORMAL_ELASTIC_STRAIN_PSD.NormalOrientation=NormalOrientationType.XAxis
NORMAL_ELASTIC_STRAIN_PSD.ScaleFactor = ScaleFactorType.Sigma1

NORMAL_STRESS_PSD = SOLUTION_PSD.AddNormalStress()
NORMAL_STRESS_PSD.NormalOrientation=NormalOrientationType.XAxis
NORMAL_STRESS_PSD.ScaleFactor = ScaleFactorType.Sigma1

SOLUTION_PSD.Activate()
RESPONSE_PSD_PROBE = SOLUTION_PSD.AddResponsePSD()
RESPONSE_PSD_PROBE.LocationMethod = LocationDefinitionMethod.CoordinateSystem
RESPONSE_PSD_PROBE.CoordinateSystemSelection=COORDINATE_SYSTEM
RESPONSE_PSD_PROBE.Reference=ResultRelativityType.Relative
RESPONSE_PSD_PROBE.ResultType = ProbeResultType.DeformationProbe
RESPONSE_PSD_PROBE.ResultSelection=ProbeDisplayFilter.XAxis

RESPONSE_PSD_PROBE2 = SOLUTION_PSD.AddResponsePSD()
RESPONSE_PSD_PROBE2.LocationMethod = LocationDefinitionMethod.RemotePoint
RESPONSE_PSD_PROBE2.RemotePointSelection = REMOTE_POINT
RESPONSE_PSD_PROBE2.Reference=ResultRelativityType.Relative
RESPONSE_PSD_PROBE2.ResultType = ProbeResultType.DeformationProbe
RESPONSE_PSD_PROBE2.ResultSelection=ProbeDisplayFilter.XAxis

SOLUTION_PSD.Activate()
SEL=ExtAPI.SelectionManager.AddSelection(NS_BLOCK3_VERTEX)
RS_PSD_TOOL = PSD.Solution.AddResponsePSDTool()
CLRSEL = ExtAPI.SelectionManager.ClearSelection()

RESPONSE_PSD_PROBE3 = RS_PSD_TOOL.Children[0]
RESPONSE_PSD_PROBE3.Name="Response PSD 3"
RESPONSE_PSD_PROBE3.Reference=ResultRelativityType.Relative
RESPONSE_PSD_PROBE3.ResultType = ProbeResultType.DeformationProbe
RESPONSE_PSD_PROBE3.ResultSelection=ProbeDisplayFilter.XAxis

# Scenario 13 Solve and review results
PSD.Activate()
PSD.Solve(True)

SOLUTION_MODAL.Activate()
FREQUENCY = MODAL.GetResultsData()

MODAL_FREQ_1ST = FREQUENCY.ListTimeFreq[0]
MODAL_FREQ_2ND = FREQUENCY.ListTimeFreq[1]
MODAL_FREQ_3RD = FREQUENCY.ListTimeFreq[2]

DIRECTIONAL_DEFORMATION_PSD.Activate()
MAX_X_AXIS_DEF_PSD = DIRECTIONAL_DEFORMATION_PSD.Maximum.Value

DIRECTIONAL_VELOCITY_PSD.Activate()

```

```
MIN_X_AXIS_VEL_PSD = DIRECTIONAL_VELOCITY_PSD.Minimum.Value
MAX_X_AXIS_VEL_PSD = DIRECTIONAL_VELOCITY_PSD.Maximum.Value

DIRECTIONAL_ACCELERATION_PSD.Activate()
MIN_X_AXIS_ACC_PSD = DIRECTIONAL_ACCELERATION_PSD.Minimum.Value
MAX_X_AXIS_ACC_PSD = DIRECTIONAL_ACCELERATION_PSD.Maximum.Value

NORMAL_ELASTIC_STRAIN_PSD.Activate()
MAX_X_AXIS_STRAIN_PSD = NORMAL_ELASTIC_STRAIN_PSD.Maximum.Value

NORMAL_STRESS_PSD.Activate()
MAX_X_AXIS_STRESS_PSD = NORMAL_STRESS_PSD.Maximum.Value

RESPONSE_PSD_PROBE.Activate()
RMS_VALUE_RES_PSD = RESPONSE_PSD_PROBE.RMSValue.Value
EXP_FREQ_RES_PSD = RESPONSE_PSD_PROBE.ExpectedFrequency.Value

RESPONSE_PSD_PROBE2.Activate()
RMS_VALUE_RES_PSD2 = RESPONSE_PSD_PROBE2.RMSValue.Value
EXP_FREQ_RES_PSD2 = RESPONSE_PSD_PROBE2.ExpectedFrequency.Value

RESPONSE_PSD_PROBE3.Activate()
RMS_VALUE_RES_PSD3 = RESPONSE_PSD_PROBE3.RMSValue.Value
EXP_FREQ_RES_PSD3 = RESPONSE_PSD_PROBE3.ExpectedFrequency.Value
```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.

Static Structural Analysis using External Model

In this example, using the support files, you will perform a Static Structural analysis using an imported mesh-based geometry from an **External Model** system. Using a sequence of python journal commands, you will automatically define and solve the analysis.

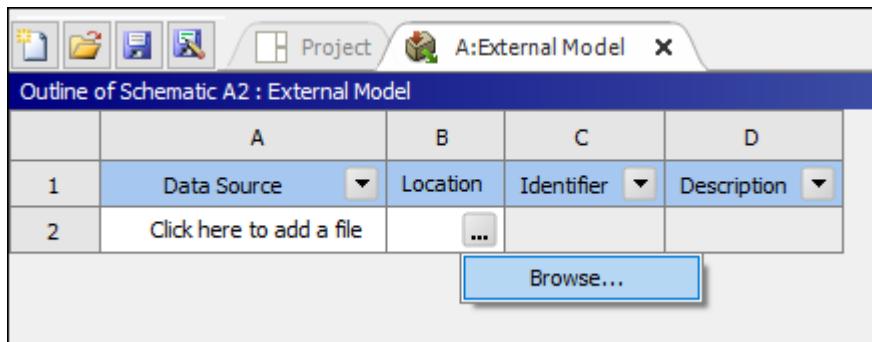
This example begins in the Mechanical application. It requires you to download the following files.

- `External_Model.bfd`
- `External_Model.py`

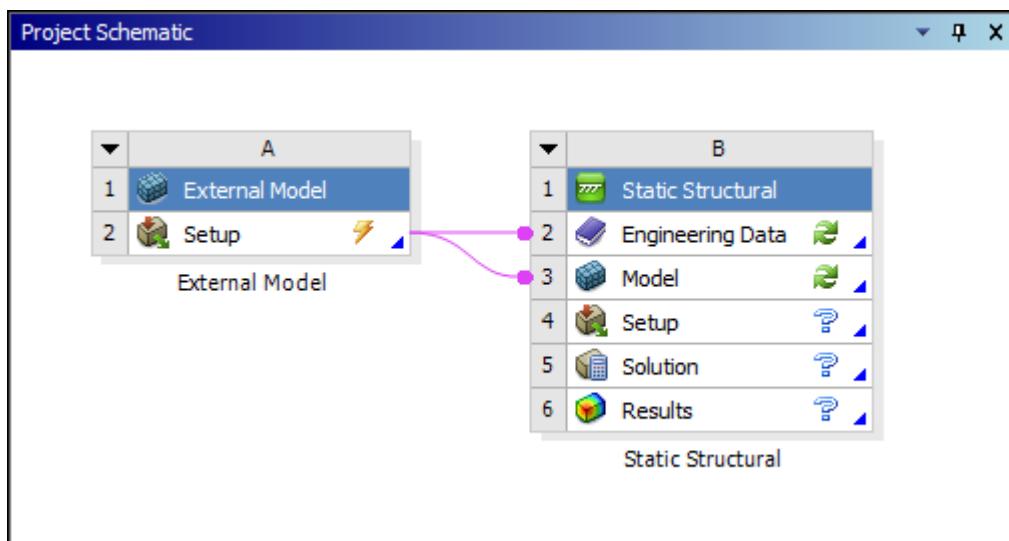
These files are available [here](#).

Procedure

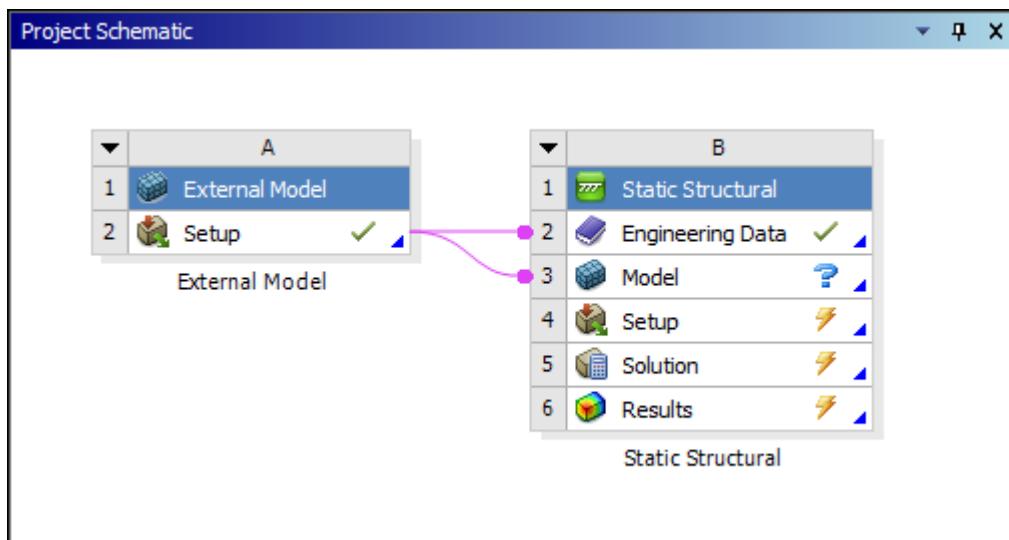
1. Open Workbench and insert an **External Model** system, from the **Component Systems** group in the **Toolbox**, into the **Project Schematic**.
2. Double-click the **Setup** cell of the **External Model** system.
3. From the **External Model** tab, select the option of the **Location** column and select **Browse**.



4. Navigate to the proper folder location and select the sample file, External_Model.bfd.
5. Return to the project page.
6. Drag and drop a **Static Structural** system onto the **Setup** cell of the **External Model** system so that it is linked as shown below.



7. Select the **Update Project** option on the toolbar.



8. Open Mechanical: right-click the **Model** cell and select **Edit**.
9. Select the **Automation tab** and select the **Scripting** option to open the **Mechanical Scripting** pane.

10. Select the **Open Script** option () from the **Editor (p. 4)** toolbar. Navigate to the proper folder location and select `External_Model.py`.

11. Select the **Run Script** option () from the **Editor (p. 4)** toolbar.

Scripts Illustrated

In this example, the python file automatically performs the following actions:

```
# Scenario 1 Set up the Tree Object Items: Define the geometry, coordinate system, connections, mesh, static system

GEOMETRY= Model.Geometry
COORDINATE_SYSTEM = Model.CoordinateSystems
CONNECTIONS = Model.Connections
MESH = Model.Mesh
STATIC_STRUCTURAL = ExtAPI.DataModel.AnalysisByName("Static Structural")
ANALYSIS_SETTINGS = STATIC_STRUCTURAL.AnalysisSettings
SOLUTION= STATIC_STRUCTURAL.Solution

# Scenario 2 Set up Unit System: Set up the Unit Systems to Metric (m, kg, N, s, V, A)

ExtAPI.Application.ActiveUnitSystem = MechanicalUnitSystem.StandardMKS

# Scenario 3 Shell Thicknesses Imported from External Model: working with Get and Set methods for getting and setting

SHELL_THICKNESSES_WORKSHEET=[x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Shell Thicknesses(External Model)']
SHELL_THICKNESSES_WORKSHEET.Activate()
SHELL_THICKNESSES_PROPERTY_NAMES= SHELL_THICKNESSES_WORKSHEET [0].GetPropertyNames()
SHELL_THICKNESS_OBJECT1_ID = SHELL_THICKNESSES_WORKSHEET [0].GetPropertyAsString("Id")
SHELL_THICKNESS_OBJECT1_SCOPE = SHELL_THICKNESSES_WORKSHEET [0].GetPropertyAsString("Scoping")
SHELL_THICKNESS_OBJECT1_THICKNESS = SHELL_THICKNESSES_WORKSHEET [0].GetPropertyAsString("Thickness")
SHELL_THICKNESS_OBJECT1_OFFSET_TYPE = SHELL_THICKNESSES_WORKSHEET [0].GetPropertyAsString("OffsetType")
SHELL_THICKNESS_OBJECT4_OFFSET = SHELL_THICKNESSES_WORKSHEET [4].GetPropertyAsString("Offset")
SHELL_THICKNESSES_WORKSHEET [4].SetPropertyValue("Offset", -0.01)

# Scenario 4 Point Masses Imported from External Model: working with Get and Set methods for getting and setting

POINT_MASSES_WORKSHEET= [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Point Masses(External Model)'][0]
POINT_MASSES_WORKSHEET.Activate()
POINT_MASSES_PROPERTY_NAMES = POINT_MASSES_WORKSHEET[0].GetPropertyNames()
POINT_MASSES_OBJECT1_ID = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("Id")
POINT_MASSES_OBJECT1_REFERENCENODEID = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("ReferenceNodeId")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("ReferenceNodeId", "54_firstNode")
POINT_MASSES_OBJECT1_REFERENCENODEID = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("ReferenceNodeId")
POINT_MASSES_OBJECT1_MASS = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("Mass")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("Mass", "1e5")
POINT_MASSES_OBJECT1_MASS = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("Mass")
POINT_MASSES_OBJECT1_INERTIAX= POINT_MASSES_WORKSHEET[0].GetPropertyAsString("InertiaX")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("InertiaX", "100")
POINT_MASSES_OBJECT1_INERTIAZ = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("InertiaX")
POINT_MASSES_OBJECT1_INERTIAY = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("InertiaY")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("InertiaY", "80")
POINT_MASSES_OBJECT1_INERTIAY = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("InertiaY")
POINT_MASSES_OBJECT1_INERTIAZ = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("InertiaZ")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("InertiaZ", "50")
POINT_MASSES_OBJECT1_INERTIAZ = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("InertiaZ")
POINT_MASSES_OBJECT1_INERTIAXY = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("InertiaXY")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("InertiaXY", "0")
POINT_MASSES_OBJECT1_INERTIAXY = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("InertiaXY")
```

```

POINT_MASSES_OBJECT1_INERTIAZ = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("InertiaXZ")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("InertiaXZ", "0")
POINT_MASSES_OBJECT1_INERTIAZ = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("InertiaXZ")
POINT_MASSES_OBJECT1_INERTIAYZ = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("InertiaYZ")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("InertiaYZ", "0")
POINT_MASSES_OBJECT1_INERTIAYZ = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("InertiaYZ")
POINT_MASSES_OBJECT1_OFFSETX = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("OffsetX")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("OffsetX", "10")
POINT_MASSES_OBJECT1_OFFSETX = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("OffsetX")
POINT_MASSES_OBJECT1_OFFSETY = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("OffsetY")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("OffsetY", "8")
POINT_MASSES_OBJECT1_OFFSETY = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("OffsetY")
POINT_MASSES_OBJECT1_OFFSETZ = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("OffsetZ")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("OffsetZ", "5")
POINT_MASSES_OBJECT1_OFFSETZ = POINT_MASSES_WORKSHEET[0].GetPropertyAsString("OffsetZ")
POINT_MASSES_WORKSHEET[1].SetPropertyValue("InertiaXY", "0")
POINT_MASSES_WORKSHEET[1].SetPropertyValue("InertiaXZ", "0")
POINT_MASSES_WORKSHEET[1].SetPropertyValue("InertiaYZ", "0")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("OffsetX", "0")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("OffsetY", "0")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("OffsetZ", "0")
POINT_MASSES_WORKSHEET[0].SetPropertyValue("Mass", "1e3")

```

Scenario 5 Rigid Remote Connectors Imported from External Model: working with Get and Set methods for getting and setting properties.

```

RIGID_REMOTE_CONNECTORS_WORKSHEET= [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Rigid Remote Connectors']
RIGID_REMOTE_CONNECTORS_WORKSHEET.Activate()
RIGID_REMOTE_CONNECTORS_PROPERTY_NAMES = RIGID_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyNames()
RIGID_REMOTE_CONNECTORS_OBJECT1_TYPE = RIGID_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("RigidConnectors")
RIGID_REMOTE_CONNECTORS_OBJECT1_ID = RIGID_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("Id")
RIGID_REMOTE_CONNECTORS_OBJECT1_REFERENCE_NODE_ID = RIGID_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("ReferenceNodeId")
RIGID_REMOTE_CONNECTORS_WORKSHEET[0].SetPropertyValue("ReferenceNodeId", "53_firstNode")
RIGID_REMOTE_CONNECTORS_OBJECT1_REFERENCE_NODE_ID = RIGID_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("ReferenceNodeId")
RIGID_REMOTE_CONNECTORS_OBJECT1_DOFS = RIGID_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("Dofs")
RIGID_REMOTE_CONNECTORS_WORKSHEET[0].SetPropertyValue("Dofs", "uz; rot x; rot y")
RIGID_REMOTE_CONNECTORS_OBJECT1_DOFS = RIGID_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("Dofs")
RIGID_REMOTE_CONNECTORS_OBJECT1_PARTICIPATING_NODES = RIGID_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("Nodes")
RIGID_REMOTE_CONNECTORS_WORKSHEET[0].SetPropertyValue("Dofs", "ux; uy; uz; rot x; rot y; rot z")

```

Scenario 6 Flexible Remote Connectors Imported from External Model: working with Get and Set methods for getting and setting properties.

```

FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET= [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Flexible Remote Connectors']
FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET.Activate()
FLEXIBLE_REMOTE_CONNECTORS_PROPERTY_NAMES = FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyNames()
FLEXIBLE_REMOTE_CONNECTORS_OBJECT1_TYPE = FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("Connectors")
FLEXIBLE_REMOTE_CONNECTORS_OBJECT1_ID = FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("Id")
FLEXIBLE_REMOTE_CONNECTORS_OBJECT1_REFERENCE_NODE_ID = FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("ReferenceNodeId")
FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].SetPropertyValue("ReferenceNode", "54_firstNode")
FLEXIBLE_REMOTE_CONNECTORS_OBJECT1_REFERENCE_NODE_ID = FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("ReferenceNodeId")
FLEXIBLE_REMOTE_CONNECTORS_OBJECT1_DOFS = FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("Dofs")
FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].SetPropertyValue("Dofs", "uz; rot x; rot y")
FLEXIBLE_REMOTE_CONNECTORS_OBJECT1_DOFS = FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("Dofs")
FLEXIBLE_REMOTE_CONNECTORS_OBJECT1_PARTICIPATING_NODES = FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("Nodes")
FLEXIBLE_REMOTE_CONNECTORS_OBJECT1_WEIGHT_PARTICIPATING_FACTOR = FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("WeightParticipatingFactor")
FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].SetPropertyValue("WeightParticipatingFactor", 2)
FLEXIBLE_REMOTE_CONNECTORS_OBJECT1_WEIGHT_PARTICIPATING_FACTOR = FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("WeightParticipatingFactor")
FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].SetPropertyValue("WeightParticipatingFactor", 1)
FLEXIBLE_REMOTE_CONNECTORS_OBJECT1_WEIGHT_PARTICIPATING_FACTOR = FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].GetPropertyAsString("WeightParticipatingFactor")
FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].SetPropertyValue("Dofs", "ux; uy; uz; rot x; rot y; rot z")

```

Scenario 7 Spring Connectors Imported from External Model: working with Get and Set methods for getting and setting properties.

```

SPRING_CONNECTORS_WORKSHEET= [x for x in ExtAPI.DataModel.Tree.AllObjects if x.Name == 'Spring Connectors(External Model)']
SPRING_CONNECTORS_WORKSHEET.Activate()
SPRING_CONNECTORS_PROPERTY_NAMES = SPRING_CONNECTORS_WORKSHEET[0].GetPropertyNames()
SPRING_CONNECTORS_OBJECT4_TYPE = SPRING_CONNECTORS_WORKSHEET[3].GetPropertyAsString("Type")
SPRING_CONNECTORS_OBJECT4_ID = SPRING_CONNECTORS_WORKSHEET[3].GetPropertyAsString("Id")
SPRING_CONNECTORS_OBJECT4_NODES = SPRING_CONNECTORS_WORKSHEET[3].GetPropertyAsString("Nodes")
SPRING_CONNECTORS_OBJECT1_STIFFNESS = SPRING_CONNECTORS_WORKSHEET[0].GetPropertyAsString("Stiffness")
SPRING_CONNECTORS_OBJECT1_DAMPING = SPRING_CONNECTORS_WORKSHEET[0].GetPropertyAsString("Damping")
SPRING_CONNECTORS_OBJECT4_COORDINATE_SYSTEM = SPRING_CONNECTORS_WORKSHEET[3].GetPropertyAsString("CoordinateSystem")

```

```
SPRING_CONNECTORS_OBJECT4_LOCATION= SPRING_CONNECTORS_WORKSHEET[3].GetPropertyAsString("Location")
SPRING_CONNECTORS_OBJECT4_LOCATION_COORDINATE_SYSTEM = SPRING_CONNECTORS_WORKSHEET[3].GetPropertyAsString("Loc
SPRING_CONNECTORS_WORKSHEET[3].SetPropertyValue("GroundedNode", 2)
SPRING_CONNECTORS_OBJECT4_GROUNDEDNODE = SPRING_CONNECTORS_WORKSHEET[3].GetPropertyAsString("GroundedNode")
SPRING_CONNECTORS_WORKSHEET[3].SetPropertyValue("GroundedNode", 0)
SPRING_CONNECTORS_OBJECT4_GROUNDEDNODE = SPRING_CONNECTORS_WORKSHEET[3].GetPropertyAsString("GroundedNode")
SPRING_CONNECTORS_WORKSHEET.Suppress = True
SPRING_CONNECTORS_WORKSHEET.Suppress = False

# Scenario 8 Define Analysis Settings: Set Number of Steps is equal to 4 under details view of Analysis Settings.

ANALYSIS_SETTINGS.Activate()
ANALYSIS_SETTINGS.NumberOfSteps=4

# Scenario 9 Insert Results: Insert a Total deformation.

SOLUTION.Activate()
TOTAL_DEFORMATION = SOLUTION.AddTotalDeformation()

# Scenario 10 Uncheck Point Masses, Rigid and Flexible connectors: Unchecking Point Mass, Rigid and Flexible connecto

POINT_MASSES_WORKSHEET.Activate()
POINT_MASSES_WORKSHEET[0].Active = False
POINT_MASSES_WORKSHEET[1].Active = False

FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET.Activate()
FLEXIBLE_REMOTE_CONNECTORS_WORKSHEET[0].Active = False

RIGID_REMOTE_CONNECTORS_WORKSHEET.Activate()
RIGID_REMOTE_CONNECTORS_WORKSHEET[0].Active = False

# Scenario 11 Solve and Review the Results: Solve the system and set the results variables. Note the Scripting windo

SOLUTION.Activate()
SOLUTION.Solve(True)

MAXIMUM_TOTAL_DEFORMATION = TOTAL_DEFORMATION.Maximum.Value
MINIMUM_TOTAL_DEFORMATION =TOTAL_DEFORMATION.Minimum.Value
```

Summary

This example demonstrates how scripting in Mechanical can be used to automate your actions.