# Table of Contents

**Scripting Overview** ................................................................. 1  
  Journaling and Scripting Capability Overview ........................................ 1  
  Journaling .................................................................................. 1  
  Recording and Playing a Journal ................................................... 2  
  Using the Command Window ....................................................... 3  
  Scripting ..................................................................................... 5  
  Command Line Execution ............................................................ 5  
  Scripting and Data-Integrated Applications ........................................ 6  
  ANSYS Workbench Project and Data Model Concepts .......................... 7  

**Using Scripting in ANSYS Workbench** ........................................ 13  
  Object-Based Approach ............................................................... 13  
  File Path Handling in ANSYS Workbench ....................................... 14  
  Units ............................................................................................ 15  
  Usage Examples .......................................................................... 16  
  Automatically Update all Projects Affected by a Design Modification 16  
  Interaction with Files in a Project .................................................. 20  
  Material Properties and Tabular Data .............................................. 23  
  Mechanical APDL and Sending Commands to Integrated Applications 25  
  Updating a Workbench Project and Parameters from Excel ............... 29  

**Known Issues and Limitations** .................................................... 33  
  Known Issues and Limitations ...................................................... 33  

**I. Data Containers** .................................................................... 35  
  Ansoft ........................................................................................... 37  
    Ansoft Feedback Iterator ............................................................ 39  
  AQWA ........................................................................................... 43  
    AQWA Model ........................................................................... 43  
    AQWA Results ......................................................................... 45  
    AQWA Setup ............................................................................ 46  
    AQWA Solution ........................................................................ 48  
  AUTODYN .................................................................................... 51  
    AUTODYN Analysis ................................................................. 51  
    AUTODYN Setup ...................................................................... 51  
  CFD Results ................................................................................ 55  
    CFD Results ............................................................................ 55  
  CFX ............................................................................................... 59  
    CFX Setup ............................................................................... 59  
    CFX Solution .......................................................................... 62  
  Design Exploration ....................................................................... 67  
    DX Direct Optimization ............................................................ 67  
    DX Evaluation Container ........................................................ 110  
    DX GDO Design of Experiment ................................................ 179  
    DX GDO Response Surface ...................................................... 200  
    DX Parameters Correlation ...................................................... 232  
    DX Six Sigma Analysis ............................................................ 249  
  Engineering Data ......................................................................... 265  
    Engineering Data ..................................................................... 265  
    Engineering Data Curve Fit ...................................................... 299  
    Engineering Data Favorite Items .............................................. 303
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engineering Data Favorite Library</td>
<td>324</td>
</tr>
<tr>
<td><strong>External Connection</strong></td>
<td>351</td>
</tr>
<tr>
<td>External Connection</td>
<td>351</td>
</tr>
<tr>
<td><strong>External Data</strong></td>
<td>353</td>
</tr>
<tr>
<td>External Data</td>
<td>353</td>
</tr>
<tr>
<td><strong>External Model Setup</strong></td>
<td>367</td>
</tr>
<tr>
<td>External Model Setup</td>
<td>367</td>
</tr>
<tr>
<td><strong>Finite Element Modeler</strong></td>
<td>375</td>
</tr>
<tr>
<td>Finite Element Modeler</td>
<td>375</td>
</tr>
<tr>
<td><strong>FLUENT</strong></td>
<td>385</td>
</tr>
<tr>
<td>FLUENT Setup</td>
<td>385</td>
</tr>
<tr>
<td>FLUENT Solution</td>
<td>398</td>
</tr>
<tr>
<td><strong>Geometry</strong></td>
<td>415</td>
</tr>
<tr>
<td>Geometry</td>
<td>415</td>
</tr>
<tr>
<td><strong>Graphics</strong></td>
<td>425</td>
</tr>
<tr>
<td>Graphics</td>
<td>425</td>
</tr>
<tr>
<td><strong>ICE</strong></td>
<td>443</td>
</tr>
<tr>
<td>ICE</td>
<td>443</td>
</tr>
<tr>
<td>ICE Setup</td>
<td>445</td>
</tr>
<tr>
<td><strong>ICEM</strong></td>
<td>449</td>
</tr>
<tr>
<td>ICEM CFD</td>
<td>449</td>
</tr>
<tr>
<td><strong>IcePak</strong></td>
<td>451</td>
</tr>
<tr>
<td>IcePak Setup</td>
<td>451</td>
</tr>
<tr>
<td>IcePak Solution</td>
<td>452</td>
</tr>
<tr>
<td><strong>Mechanical APDL</strong></td>
<td>455</td>
</tr>
<tr>
<td>Mechanical APDL</td>
<td>455</td>
</tr>
<tr>
<td><strong>Mechanical</strong></td>
<td>461</td>
</tr>
<tr>
<td>Mechanical Enhanced Model</td>
<td>461</td>
</tr>
<tr>
<td>Mechanical Model</td>
<td>462</td>
</tr>
<tr>
<td>Mechanical Results</td>
<td>471</td>
</tr>
<tr>
<td>Mechanical Setup</td>
<td>472</td>
</tr>
<tr>
<td>Mechanical Solution</td>
<td>475</td>
</tr>
<tr>
<td><strong>Mesh</strong></td>
<td>481</td>
</tr>
<tr>
<td>Mesh</td>
<td>481</td>
</tr>
<tr>
<td><strong>Microsoft Office Excel Analysis</strong></td>
<td>489</td>
</tr>
<tr>
<td>Microsoft Office Excel Analysis</td>
<td>489</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td>497</td>
</tr>
<tr>
<td>Parameters</td>
<td>497</td>
</tr>
<tr>
<td><strong>Polyflow</strong></td>
<td>507</td>
</tr>
<tr>
<td>Polyflow Setup</td>
<td>507</td>
</tr>
<tr>
<td>Polyflow Solution</td>
<td>511</td>
</tr>
<tr>
<td><strong>Project</strong></td>
<td>515</td>
</tr>
<tr>
<td>Project</td>
<td>515</td>
</tr>
<tr>
<td>Project File Types</td>
<td>526</td>
</tr>
<tr>
<td>Project Files</td>
<td>527</td>
</tr>
<tr>
<td>Project Messages</td>
<td>532</td>
</tr>
<tr>
<td><strong>System Coupling</strong></td>
<td>535</td>
</tr>
<tr>
<td>System Coupling Setup</td>
<td>535</td>
</tr>
<tr>
<td>System Coupling Solution</td>
<td>553</td>
</tr>
<tr>
<td><strong>TurboSystems</strong></td>
<td>563</td>
</tr>
<tr>
<td>Turbo Geometry</td>
<td>563</td>
</tr>
<tr>
<td>Turbo Mesh</td>
<td>568</td>
</tr>
</tbody>
</table>
Vista AFD Analysis ................................................................. 572
Vista AFD Design ................................................................. 589
Vista AFD Meanline .............................................................. 605
Vista CCD .............................................................................. 614
Vista CCM .............................................................................. 641
Vista CPD .............................................................................. 648
Vista RTD .............................................................................. 666
Vista TF Setup ....................................................................... 684
Vista TF Solution .................................................................... 692

Units ......................................................................................... 693

Namespaced Commands ....................................................... 697
  Ansoft .................................................................................. 697
  EKM .................................................................................. 700
  EngData .............................................................................. 705
  Extensions ........................................................................... 706
  Graphics ............................................................................. 709
  Mechanical .......................................................................... 712
  Meshing ............................................................................. 712
  Parameters .......................................................................... 713
  Project .................................................................................. 720

Data Types ................................................................................ 753
  Data Types ........................................................................... 753

Index ......................................................................................... 793
Scripting Overview

Journaling and Scripting Capability Overview

ANSYS Workbench offers the ability to record the actions you perform via the GUI, which we refer to as journaling. Journals are recorded as Python-based scripts. You can modify these scripts or create new ones, which we refer to as scripting. Together, these capabilities allow you to quickly and easily replay analyses you've already run via recorded journals, as well as to extend functionality, automate repetitive analyses, and run analyses in batch mode.

**Note**

Not all actions are journaled. Only actions that change project data are journaled. Some examples of actions that are not journaled include:

- GUI-only actions, such as:
  - Interrupting a Solve operation
  - Running in Compact mode
  - Launching help (including Quick Help and Sidebar Help)
  - Running the View Solver Output option from VistaTF’s Solution cell
- Actions taken in some data-integrated applications; see Scripting and Data-Integrated Applications (p. 6)

**Journaling**

A journal is a record of all operations that have modified data during your session. Based on your Preferences setting, a journal of your full session will automatically be saved to the location you specify (see Setting Journaling Preferences (p. 1)). You can also choose to record part of a session to a journal file, capturing a specific set of actions. Playing back the journal will recreate the recorded actions exactly. Journaling and scripting tools (including recording and playback) are available through the File > Scripting menu in ANSYS Workbench.

Journaled sessions can be used to restore work after a crash. Journals are platform independent and portable, subject to file location consistency between accounts (see File Path Handling in ANSYS Workbench for details on file path handling within journals and scripts). They can be used with any ANSYS installation (release 12.1 or higher).

**Setting Journaling Preferences**

You can set journaling preferences such as the default directory where journals will be written and how long to keep a journal file.
1. In ANSYS Workbench, select **Tools > Options > Journals and Logs.**

2. Select **Record Journal Files** to have ANSYS Workbench automatically write journal files.

3. Specify the default location where journal files will be written. This is the location that the file browser will open in automatically when you choose to begin recording a journal. You will still be able to browse to a different location before saving a particular journal.

4. Specify the number of days to keep a journal file.

5. Specify how long (in seconds) to pause between each command when running a journal file.

6. Click **OK** to save your settings.

**Recording and Playing a Journal**

Follow the steps described below to record a journal and then to playback a journal interactively. To use the command window, see **Using the Command Window (p. 3).**

**Recording a Journal**

1. Launch ANSYS Workbench.

2. Select **File > Scripting > Record Journal.**

3. Specify the name and location of the journal file and click **Save.**

4. Use the GUI to work through your analysis as you normally would.

5. Select **File > Scripting > Stop Recording Journal.**

6. A message appears informing you that you will stop recording. Click **OK.**

---

**Note**

Not all actions are journaled. Only actions that change project data are journaled. Some examples of actions that are not journaled include:

- GUI-only actions, such as:
  - Interrupting a Solve operation
  - Running in Compact mode
  - Launching help (including Quick Help and Sidebar Help)
  - Running the View Solver Output option from VistaTF’s Solution cell
- Actions taken in some data-integrated applications; see **Scripting and Data-Integrated Applications** (p. 6)

---

**Playing Back a Recorded Journal**

1. Select **File > Scripting > Run Script File.**
2. Select the journal file to be played back and click **Open**.

3. The previously recorded actions will be performed again.

---

**Note**

Because variables may be changed or overwritten, the results may be different on repeated runs as they will be acting on the outcome of the previous runs.

---

**Using the Command Window**

The command window allows you to invoke commands, access data entity properties, and invoke data entity and data container methods interactively, one at a time.

1. Select **File > Scripting > Open Command Window**.

2. Enter the commands you want to run, one at a time.

3. As you enter each command, the appropriate action will occur in the ANSYS Workbench GUI.

**Command Window Usage**

While recording a journal, ANSYS Workbench creates a number of variables for the object references containing the data in your project. For example, consider the following journal snippet:

```plaintext
template1 = GetTemplate(TemplateName="Thermal")
system1 = template1.CreateSystem()
```

In this journal, `template1` and `system1` are the variables for the references to the associated data objects. These variables are created and recorded specifically for replaying the journal, and they are not immediately accessible from within the command window. However, when working in the command window, you may wish to use these variables. Doing so can aid in manually examining the details of your project or assist in creating scripts based on the journal. To use these variables, execute the command `ImportJournalVariables()` in the command window to make the variable definitions from the currently-recorded journal available in the command window. You should be aware of the following points when using the `ImportJournalVariables()` command:

1. The variable definitions are based on those in the currently-recorded journal. By default, this journal is the automatically-recorded journal controlled by user preferences (see Setting Journaling Preferences (p. 1)). If you have manually started a journal recording of part of your session (see Recording a Journal (p. 2)), the variable definitions will be taken from the manually-recorded journal.

2. If you have any manually-defined variables of the same name as any journal variables, your variables will be overwritten by the journal variables.

3. Changing the definition of a journal variable in the command window after executing `ImportJournalVariables()` does not affect the definition of the variable in the currently-recorded journal.

4. The `ImportJournalVariables()` command can be executed multiple times in a session and will update the variables based on the currently-recorded journal.
Command Window Navigation

The command window uses the Python programming language to interpret and invoke commands or other operations. In addition, you can use numerous keyboard shortcuts to facilitate your window interaction.

Text Cursor Keyboard Keys  When typing a command or statement, the following special keys are available for moving the text cursor:

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Arrow</td>
<td>Moves the cursor back one character</td>
</tr>
<tr>
<td>Right Arrow</td>
<td>Moves the cursor forward one character</td>
</tr>
<tr>
<td>Ctrl + Left Arrow</td>
<td>Moves the cursor back one word</td>
</tr>
<tr>
<td>Ctrl + Right Arrow</td>
<td>Moves the cursor forward one word</td>
</tr>
<tr>
<td>Home</td>
<td>Moves the cursor to the beginning of the line</td>
</tr>
<tr>
<td>End</td>
<td>Moves the cursor to the end of the line</td>
</tr>
</tbody>
</table>

Copy/Paste Keyboard Keys  You can copy text from the command window to the clipboard or paste text from the clipboard as input to the command window. The following keys allow you to copy and paste text:

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+C/Insert</td>
<td>Copies selected text from the command window to the clipboard. Text copied from the command window is first selected (highlighted) using the mouse.</td>
</tr>
<tr>
<td>Ctrl+V/Shift+Insert</td>
<td>Pastes the text found on the clipboard into the input area of the command window. If multiple lines of text are pasted, the lines must be one or more complete Python statements.</td>
</tr>
</tbody>
</table>

Command History  The command window maintains a history of commands or statements that you enter so you can easily recall a previously-entered command or statement and invoke it again without retyping it. You could also make some modifications to it before invoking it again.

The following keys allow you to access the command history:

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up Arrow/Page Up</td>
<td>Recalls the previously-entered command, and the command before that if the key is pressed again</td>
</tr>
<tr>
<td>Down Arrow/Page Down</td>
<td>Recalls the next command in the history list</td>
</tr>
</tbody>
</table>

Command Completion  The command window provides a command-completion (tab-completion) feature to automatically complete partially-typed variables(commands to save tedious typing.

Type one or more characters and press the Tab key once or multiple times to see the defined variables and commands that have names beginning with the characters you typed. Entering an object variable name with the dot (.) and then pressing the Tab key will cycle through the defined properties and methods for that object. Entering one or more characters after the dot will restrict the completion results to just those properties and methods that start with those characters.
The following keys allow you to access the command completion:

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tab</strong></td>
<td>Completes the current text on the command line with any variable, property, command, or method name that is valid in the current context. Press <strong>Tab</strong> repeatedly to cycle forward through possible completions, if available.</td>
</tr>
<tr>
<td><strong>Shift+Tab</strong></td>
<td>Same as <strong>Tab</strong> but cycles backwards through possible completions.</td>
</tr>
</tbody>
</table>

**Scripting**

A script is a set of instructions to be issued to ANSYS Workbench. The script can be a modified journal, or it can be a completely new set of instructions that you write directly.

ANSYS Workbench uses an object-based approach; therefore, for scripting, some knowledge of object oriented programming and the Python language is advantageous. For detailed information on using Scripting, see "Using Scripting in ANSYS Workbench" (p. 13).

ANSYS Workbench scripting is based on IronPython 2.6. Before attempting to write or modify ANSYS Workbench scripts, you should be familiar with using this version of Python.

IronPython is well integrated with the rest of the .NET Framework (on Windows) and Mono CLR (on Linux) and makes all related libraries easily available to Python programmers while maintaining compatibility with the Python language. For more information on IronPython, see [http://ironpython.codeplex.com/](http://ironpython.codeplex.com/).

IronPython is generally compatible with existing standard Python scripts. However, not all C-based Python library modules are available under IronPython, as discussed on the IronPython website.

For more information on Python, including a standard language reference, see [http://www.python.org/](http://www.python.org/).

For a complete list of our published data containers, namespaced commands, and data types, see the reference material later in this document. You can also find this guide in a PDF version on www.ansys.com. Go to the Customer Portal, and after logging on, select Product Documentation and click the Workbench tab.

**Command Line Execution**

ANSYS Workbench can be executed from the operating system command line and accepts a number of command line arguments to facilitate automation and the replay of scripts. The following command can be used to run ANSYS Workbench from the command line:

```
<installation path>/v150/Framework/bin/<platform>/runwb2
```

`<platform>` will be one of the following:

- Win32
- Win64
- Linux64

For example, to run ANSYS Workbench from the default installation location on a Windows 64-bit system, the command would be:

```
C:\Program Files\Ansys Inc\V150\Framework\bin\win64\runwb2
```
The following table describes the command line arguments that can be used to control ANSYS Workbench file operations and execution behavior at start-up.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-B</td>
<td>Run ANSYS Workbench in batch mode. In this mode, the user interface is not displayed and a console window is opened. The functionality of the console window is the same as the ANSYS Workbench Command Window.</td>
</tr>
<tr>
<td>-R &lt;ANSYS Workbench script file&gt;</td>
<td>Replay the specified ANSYS Workbench script file on start-up. If specified in conjunction with -B, ANSYS Workbench will start in batch mode, execute the specified script, and shut down at the completion of script execution.</td>
</tr>
<tr>
<td>-I</td>
<td>Run ANSYS Workbench in interactive mode. This is typically the default, but if specified in conjunction with -B, both the user interface and console window are opened.</td>
</tr>
<tr>
<td>-X</td>
<td>Run ANSYS Workbench interactively and then exit upon completion of script execution. Typically used in conjunction with -R.</td>
</tr>
<tr>
<td>-F &lt;ANSYS Workbench project file&gt;</td>
<td>Load the specified ANSYS Workbench project file on start-up.</td>
</tr>
<tr>
<td>-E &lt;command&gt;</td>
<td>Execute the specified ANSYS Workbench scripting command on start-up. You can issue multiple commands, separated with a semicolon (;), or specify this argument multiple times and the commands will be executed in order.</td>
</tr>
</tbody>
</table>

The Console Window  The console window is the same as the command window but is present when running in batch mode to provide a way of working directly with commands outside of the user interface.

Scripting and Data-Integrated Applications

From the Project Schematic, you can interact with applications that are native to ANSYS Workbench (called workspaces), and you can launch applications that are data-integrated. Native workspaces are built entirely on the ANSYS Workbench framework and can use all of its services. Examples of native workspaces include the Project Schematic, Engineering Data, and Design Exploration.

Data-integrated applications are created independently of the ANSYS Workbench framework but have been extended so they can be driven by the Project Schematic and share key data and parameters with ANSYS Workbench and ANSYS Workbench-enabled applications. Data-integrated applications include the Mechanical APDL application, ANSYS Fluent, ANSYS CFX, DesignModeler, and the Mechanical application.

The difference between native workspaces and data-integrated applications is an important consideration for ANSYS Workbench journaling and scripting. All operations that modify the data associated with a native workspace are journaled and can be fully automated with ANSYS Workbench scripting. For data-integrated applications, only those operations initiated from the Project Schematic are journaled, e.g., system updates and data transfers. Operations performed within data-integrated application are not necessarily journaled or controlled by ANSYS Workbench scripting. For example, steps to construct geometry in Mechanical APDL or solution methods in ANSYS Fluent are not journaled.

Although data-integrated applications do not fully support ANSYS Workbench scripting, many of them have their own native scripting language which is accessible through the ANSYS Workbench scripting interface (see Table 1: Scripting Support for Data-Integrated Applications (p. 7)). For example, Mechanical APDL is based on the powerful ANSYS Parametric Design Language (APDL), and APDL commands...
can be directly incorporated within an ANSYS Workbench script. Use SendCommand to pass native scripting commands to data-integrated applications.

You can insert SendCommand calls into your ANSYS Workbench scripts to drive data-integrated applications; however, data-integrated applications do not necessarily record operations in the ANSYS Workbench journal. Most scriptable data-integrated applications have an independent journal where native commands are recorded.

A Yes in the Supports Scripting with SendCommand column in Table 1: Scripting Support for Data-Integrated Applications (p. 7) indicates that the data-integrated application can be driven from an ANSYS Workbench script by manually inserting SendCommand calls.

A Yes in the Supports Journaling with SendCommand column indicates that the data-integrated application records its operations with SendCommand, and that the state of the application can be restored from the ANSYS Workbench journal itself. To learn more about SendCommand, see Mechanical APDL and Sending Commands to Integrated Applications (p. 25) and the detailed description of the method in the Data Containers section of this guide.

Table 1: Scripting Support for Data-Integrated Applications

<table>
<thead>
<tr>
<th>Data-Integrated Applications</th>
<th>Native Scripting Language</th>
<th>Supports Scripting with SendCommand</th>
<th>Supports Journaling with SendCommand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mechanical APDL</td>
<td>APDL</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Mechanical</td>
<td>JScript</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CFX</td>
<td>CCL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Fluent</td>
<td>Scheme</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Aqwa</td>
<td>JScript</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Autody</td>
<td>Not Available</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CFD-Post</td>
<td>CCL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>FE Modeler</td>
<td>JScript</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DesignModeler</td>
<td>JScript</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Meshing</td>
<td>JScript</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Polyflow</td>
<td>Not Available</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IcePak</td>
<td>Not Available</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICEM CFD</td>
<td>TCL</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

ANSYS Workbench Project and Data Model Concepts

Project Elements

You should understand the following terms and concepts when working with ANSYS Workbench scripting. The following graphic depicts the relationship between the objects in a project.
**Project,**  
The project is the full collection of systems, components, data, and their connections that you create to achieve an overall CAE goal.

**System,**  
A system is a collection of components that together provide workflow to achieve an engineering simulation goal, such as completing a static structural analysis or a steady state fluid flow simulation. Systems may also contain a single component to complete an analysis sub-task, for example, meshing a geometric assembly.

**Component,**  
A component includes a collection of data and a data editor that work together to achieve a CAE-related task. A system may include several components which together to define a simulation workflow. A data editor can be an ANSYS Workbench-enabled application like ANSYS Fluent or Mechanical, or a native ANSYS Workbench workspace like Engineering Data or Design Exploration. In the ANSYS Workbench user interface, a component is depicted as a cell in the Project Schematic.

In an abstract sense, a component receives data from external or upstream sources, enables you to edit data local to the container, and then performs operations to produce output data. For example, a meshing component may receive geometry data from an upstream source, enable you define meshing attributes, and then generate a mesh for a downstream analysis.

In addition to managing the data and data editor required for a CAE task, a component also provides common services to control the input and output data from the component. For example, the component allows its output data to be transferred to other components.

**Data Container,**  
A data container includes data unique to an individual component as well as the services to manage and manipulate it. Although most components have a single data container, they may have more than one.

Services which create, retrieve, or modify a component’s local data are provided by the data container, while services to control the transfer of data into and out of the component are provide by the component.
Data Entity,
A data entity is a data structure defined within the data container. A data container often employs several data entities. A data entity is similar to a class in an object-oriented programming language; it defines member data (or properties) and methods which can be called to perform operations on the data.

Data Reference,
A data reference is a handle to an instantiated data entity. From a data entity definition, an object can be created, and the data reference provides access to that object. In an ANSYS Workbench journal, data references are assigned to variables. The variables are then used to access the properties and methods of the data entity. Consider the following example:

```python
system = GetSystem(Name="My Analysis")
system.Update()
```

The first line queries for a previously instantiated System data entity named “My Analysis.” The `GetSystem` method returns a data reference to the specified System object and assigns it to the variable `system`. Using this variable, the second line calls a method supported by System data entity called `Update`. This method updates the state of the system referenced by the variable, namely “My Analysis.”

Data Container Reference,
Similar to data references, a data container reference is a handle to an instantiated data container object.

Scripting Interface Concepts

Object,
An object encapsulates a combination of data and services that act on the data. In the ANSYS Workbench scripting interface, data entities, data containers, or components are all examples of objects. Access to these objects is provided by data references and data container references.

Property,
A object’s data is referred to as its properties. In the ANSYS Workbench scripting interface, only objects derived from data entities have properties. A property is defined by its name, type, and value. Property types include:

• Boolean
• String
• Numerical types (Integer, Real)
• Physical quantities
• Data references and data container references
• Collections of the above types (Lists, Dictionaries)

An object’s properties are accessed by applying the dot operator on the corresponding reference to that object. In the example below, `parameter1` is a data reference to a Parameter object, and the Expression property of that object is set to 10.

```python
parameter1.Expression = "10"
```

Method,
A method is a command that is bound to a data entity or data container object. Methods can change property values, create or delete properties, or invoke complex calculations and operations. Methods
may require arguments to further specify the action to be taken, and they may return an object as a result of that action. Like properties, an object’s methods are accessed by applying the dot operator on a corresponding data reference. In the following example, `parameter1` is a data reference to a Parameter object, and the method `SetQuantityUnits` is called to set the object’s units to meters.

```plaintext
parameter1.SetQuantityUnits("m")
```

**Argument,**

An argument is a variable that can be passed to a method when invoked. An argument has a type and share most of the same types as properties.

ANSYS Workbench methods use ‘Named’ arguments, where each argument is given a name that is used to distinguish it from other arguments in the method. Because the arguments are named, their order is inconsequential.

Arguments can be required or optional. All required arguments must be specified or the method will fail. Optional arguments may be omitted and typically take a default value.

In the following example, the density property of a material is set to 8500 kg/m\(^3\). The named arguments are “Variables” and “Values” which are set to “Density” and “8500 [kg m\(^{-3}\)]”, respectively.

```plaintext
property1.SetData(Variables="Density", Values="8500 [kg m\(^{-3}\)]")
```

Because ANSYS Workbench uses named arguments, the following form of the command is also acceptable.

```plaintext
property1.SetData(Values="8500 [kg m\(^{-3}\)]", Variables="Density")
```

**Query,**

A query is a method which typically returns a data or container reference which can then be used to further interrogate or modify the project. Queries by themselves do not change the state of the project. Like methods, queries may require specifying arguments.

**Namespaced Commands,**

Namespaced commands are commands that are not bound to a particular object but have been grouped in a namespace with other commands having similar context. Like methods, commands perform some action, often require arguments, and may return an object as a result of the action. In the following example, the Update command is called in the Project namespace to update all components in a project by refreshing the input data and performing local calculations:

```plaintext
Project.Update()
```

**Templates**

ANSYS Workbench uses templates to create the Projects, System, and Components elements described above. A template is a high-level description of the item to be created, but does not contain specific detailed data.

Templates are analogous to document templates provided for Microsoft Word. For example, you may pick a Word report template that contains the formatting, sections, etc., for a type of report you wish to create. You then create a document from that template and fill out your specific content within that document to produce the report. Comparably, you may pick a System Template in ANSYS Workbench that describes the components, data, and relationships needed to execute a particular type of CAE analysis. You then create a system from that template, enter data, and perform operations within that system to complete the analysis.
Templates facilitate and automate creation of project elements for a specific purpose. However, these templates do not preclude you from manually (and flexibly) building up a project from the constituent elements to achieve the same goal.

**Project Template**
A project template defines a set of system templates and the connections between them that can be used to perform a multi-disciplinary CAE analysis task (e.g., Thermal-Stress or One-Way FSI).

The project template can be used to create specific instances of systems and components within the project (such as the Custom Templates in ANSYS Workbench Toolbox).

**System Template**
A system template contains the information to create an ANSYS Workbench System designed for a particular simulation objective. ANSYS Workbench provides system templates for many types of standard analyses, including static structural, fluid flow, explicit dynamics, steady-state thermal, and others. The Analysis Systems listed in the ANSYS Workbench Toolbox are all examples of system templates. All analyses performed in ANSYS Workbench begin by referencing a system template.

**Component Template**
The component template includes the allowed input/output data types, internal data, and key commands associated with a specific component (e.g., Geometry).
Using Scripting in ANSYS Workbench

Object-Based Approach

ANSYS Workbench scripting follows an object-based approach, where interaction with data is defined in terms of objects. Objects have properties that can be used to get or set data values, and methods that use or modify the data. In terms of the ANSYS Workbench concepts described in ANSYS Workbench Project and Data Model Concepts (p. 7), objects are references to data entities in the data model, and the methods are commands and queries which operate on those entities.

For example, a Parameter is a data entity that has properties such as description, value, and expression, and has methods that operate on the parameter to (for example) delete it or determine which other parts of the data model are associated with the parameter.

ANSYS Workbench objects fall into two general categories: data containers and data entities. The basic process for operating on these objects is:

1. **Query for either a data container or data entity objects.** Queries are implemented as methods on both data container and data entity objects, and they most often start with Get (e.g., GetFiles, GetSystem, GetComponent). These methods return references to objects that are assigned to variables. The variables can then be used to access object properties and methods. To illustrate this process, consider the Design of Experiments (DOE) data container that includes a data entity for the DOE model, which in turn contains a data entity for the input parameters. To query the input parameter object, first query the DOE model from the data container, and then query the input parameter from the DOE model:

```python
DOEModel = DOEDataContainer.GetModel()
InputParameter = DOEModel.GetParameter(Name="P1")
```

In most instances, the variable for a data reference will be reused later in the journal or script so that queries do not need to be re-executed. However, there are some instances (e.g., System Coupling Variables), where re-executing the query can clarify the context that is used to access the object. In those situations, journal variables are not reused and queries are generated each time an object is referenced.

2. **Interrogate and modify object properties.** If the query returns a reference to a data entity, you can interrogate its properties and modify those that are not identified as “Read-Only” in the Reference section of this guide. Properties are accessed by appending a dot and the property name to the variable assigned to the object reference. For example, once a reference to an input parameter is obtained, you can modify its classification (or Nature Property) to reflect that it is a continuous or discrete parameter.

```python
inputParameter.Nature = "NatureContinuous"
```

3. **Call methods on objects.** In addition to properties, most objects provide methods that operate on internal data. To call a method on an object, append a dot, the method name, and a comma-separated argument list in closed parentheses. A method's required and optional arguments are also documented in the Reference section of this guide. Continuing the input parameter example, you can specify a restricted set of Manufacturable Values for a parameter by calling the AddLevels method on the input parameter object and by constructing a list of values and assigning it to the Levels argument.
inputParameter.AddLevels( Levels=["65", "70", "75", "80"])

Typical usage examples are provided below. These examples demonstrate how to query objects and how to invoke methods on those objects for the desired result. Use the Reference section at the end of this document for a complete list of all available data container objects, and the data entities, methods, properties, and arguments associated with each.

File Path Handling in ANSYS Workbench

ANSYS Workbench uses specific conventions for file or directory paths that are recorded to an ANSYS Workbench journal, thus improving the portability of these paths between operating systems and user directories.

Handling Slashes as File Path Separators

ANSYS Workbench uses the following conventions when a forward slash or a backslash is used as a path separator in a journal or script:

1. When a journal is written, all backslashes that occur in command arguments representing a path are converted to forward slashes.
2. When a command argument containing a path is read, all slashes are internally converted to the current platform-appropriate slash.

Handling of Absolute/Relative Paths

To ensure a robust recording and playback of scripts and journals, ANSYS Workbench always records file and directory paths using a full (absolute) path. However, to facilitate portability of journals to different locations on a file system, a User Path Root is used to dynamically record and construct the absolute path.

The following conventions and capabilities are used with the User Path Root.

1. The default value of the User Path Root is taken from the Default Folder for Permanent Files preference. Changing this preference will take effect in your next session.
2. When a path is recorded to the journal and the start of the path matches the User Path Root, then the path is recorded using AbsUserPathName("<relative path name>"). This function constructs an absolute path based on the supplied relative path and the current User Path Root.
3. You can access or change the current value of User Path Root within a script using the following commands:

   • pathRoot = GetUserPathRoot()

   • SetUserPathRoot(DirectoryPath="<new path root>")

   SetUserPathRoot does not change the Default Folder for Permanent Files preference, but simply overrides the path root setting in the current session.

Example

If the current value of the User Path Root is C:\Users\myUser\Projects, then the command:
Open(FilePath=r"C:\Users\myUser\Projects\proj1.wbpj")

would be journaled as:

Open(FilePath=AbsUserPathName("proj1.wbpj"))

You can override the current User Path Root to control the base location of files in your script:

SetUserPathRoot(DirectoryPath = "C:/Users/myUser1/Projects")
Open(FilePath=AbsUserPathName("proj1.wbpj")) # Read project from first location
SetUserPathRoot(DirectoryPath = "C:/Users/myUser2/Projects")
Open(FilePath=AbsUserPathName("proj1.wbpj")) # Read project from second location

### Units

#### Specifying Quantities Without Units

Many properties and variable values in ANSYS Workbench represent physical quantities, which include both a value and a unit in their definition (see Expressions, Quantities, and Units in the Workbench User's Guide). The assignments of these quantities are journaled using a “Value [Unit]” string syntax. This method ensures that all available information is recorded in the journal. However, strings may be inconvenient to work with when writing or modifying scripts that assign quantities.

As a convenience, ANSYS Workbench allows the assignment of a quantity using just the numeric value. When units are omitted, the unit is assumed to be the unit for that quantity in the current project unit system. Although this method is more convenient, you must ensure that the value being supplied is consistent with the current units.

**Setting an Entity Property**  When setting an entity property that refers to a quantity, the property assignment can be done as a numeric value, and the units will be taken from the project unit system. For example, after setting the Inlet Mass Flow in a VistaTF setup, the following would be recorded in the journal:

```py
vistaTFSetup1 = setup1.GetSetupEntity()
vistaTFSetup1.MassFlow = "0.5 [kg s^-1]"
```

When entering this command or writing script, you can use a direct numeric value:

```py
vistaTFSetup1 = setup1.GetSetupEntity()
SetProjectUnitSystem(UnitSystemName="SI")
vistaTFSetup1.MassFlow = "0.3 [lbm s^-1]" # Units explicitly provided
print vistaTFSetup1.MassFlow
>>> 0.3 [lbm s^-1]
vistaTFSetup1.MassFlow = 0.3 # Units are taken from the project unit system
print vistaTFSetup1.MassFlow
>>> 0.3 [kg s^-1]
```

**Setting Quantity in Variable Data Tables**  The same principles apply when setting variables in Material Property data tables (used primarily in Engineering Data). For example, after selecting a material and changing the Density to 9000 [kg m^-3], the following would be recorded in the journal:

```py
material1= eda1.GetMaterial(Name="Structural Steel")
materailProperty1= material1.GetProperty(Name="Density")
materailProperty1.SetData{
    SheetName="Density",
    Variables=["Density"],
    Values=[["9000 [kg m^-3]"]])
```

When writing a script, for convenience, you can omit the units, and they will be taken from the current project unit system. You can also omit the list brackets because only single values are being specified. A condensed version of the above command that is valid when playing back a script is:
A more complex example showing the creation of a temperature-dependant property using a script is given below:

```python
# Temperatures in degrees Fahrenheit
temperatures = [200, 400, 600, 800, 1000]
# Coefficient of Thermal Expansion in F^-1
alphas = [6.3e-6, 7.0e-6, 7.46e-6, 7.8e-6, 8.04e-4]

# Change to an appropriate unit system
#(US Customary, which has an internal tag of "BIN_STANDARD")
SetProjectUnitSystem(UnitSystemName="BIN_STANDARD")

# Create a new instance of engineering data and
# access the Coefficient of Thermal Expansion property of Structural Steel
EDAtemplate = GetTemplate(TemplateName="EngData")
system = EDAtemplate.CreateSystem()
eda = system.GetContainer(ComponentName="Engineering Data")
steel = eda.GetMaterial(Name="Structural Steel")
alpha = steel.GetProperty(Name="Coefficient of Thermal Expansion")

# Set the property data according to the provided data
alpha.SetData(Variables=["Temperature","Coefficient of Thermal Expansion"],
              Values=[temperatures, alphas])
```

### Usage Examples

Several examples are provided here to demonstrate some typical scripting uses:

- **Automatically Update all Projects Affected by a Design Modification**
- **Interaction with Files in a Project**
- **Material Properties and Tabular Data**
- **Mechanical APDL and Sending Commands to Integrated Applications**
- **Updating a Workbench Project and Parameters from Excel**

### Automatically Update all Projects Affected by a Design Modification

You have performed a set of analyses on a design with an initial geometry file. These analyses have been saved as a number of different ANSYS Workbench projects. The design has been modified, and you have been provided with a new geometry file that represents the updated design.

To automate the update of all affected projects, you would like to write a script that does the following:

1. Finds all ANSYS Workbench projects within a specified directory.
2. For each project that has been found:
   - Replaces the original geometry with the new geometry file in any system in the project.
   - Updates the project and reports any errors from the update.
   - If the update was successful, reports the values of key output parameters under the modified geometry.
   - Saves the modified project with the new geometry file to a new directory.

Although the analysis specifics are secondary for the purposes of this example, we will use a CFD analysis of a blood mixing device. This device attempts to maximize mixing of two blood streams while...
minimizing flow vorticity (which is an indicator of blood damage potential). Three projects involving a coarse mesh model, a fine mesh model, and an asymmetric flow condition were created. The parameters of interest are pressure drop, average and maximum vorticity, and mixing factor of blood stream 1 at the exit.

Figure 1: Mixing in base geometry

![Mixing in base geometry](image)

Figure 2: Mixing in modified geometry

![Mixing in modified geometry](image)

**Workbench Script** The script for this example follows. Each line is numbered for reference in the discussion that follows.

```python
1 # import the 'os' module, which provides a portable way of using operating system-dependent functionality
2 import os
3
4 # Helper function to write parameters to the log file
5 def writeParams(logFile):
6    for param in Parameters.GetAllParameters():
7        prmString = " " + param.Name + ": " + param.DisplayText + " = " +
8        logFile.write(prmString + "\n")
9    logFile.flush()
10
11 # Define the original and target directories
12 origDir = AbsUserPathName("Demo/ScriptExample1/Original")
13 newDir = AbsUserPathName("Demo/ScriptExample1/Modified")
14
15 # Define new geometry file
16 newGeom = AbsUserPathName("Demo/ScriptExample1/Geometry/bloodMix2.agdb")
17
18 # Open a log file to record script progress
19 open(AbsUserPathName("Demo/ScriptExample1/GeometryReplace.log"),"w")
20
21 # Create the new directory if necessary
22 if not os.path.exists(newDir):
23    os.makedirs(newDir)
24
25 # Find all the projects in the original directory
26 projList = []
27 for fileName in os.listdir(origDir):
```
if fileName.endswith(".wbpj"):
    projList.append(fileName)

# Process each project one at a time
for projFile in projList:

    # Open the project into Workbench and clear any starting messages
    projPath = os.path.join(origDir,projFile)
    logFile.write("Processing project in " + projPath + "\n")
    Open(FilePath=projPath)
    ClearMessages()

    # Output project parameter values before update
    logFile.write("Parameter values in original project:\n")
    writeParams(logFile)

    # Walk through each system in the project and replace the geometry
    file (if there is a geometry Component)
    for system in GetAllSystems():
        try:
            geometry = system.GetContainer(ComponentName="Geometry")
        except:
            logFile.write("No geometry to replace in system " +
            system.DisplayText + "\n")
        else:
            geometry.SetFile(FilePath=newGeom)

    # Update the project
    Update()

    # If the project has been sucessfully updated, write out the new
    parameter values
    if IsProjectUpToDate():
        logFile.write("Parameter values in revised project:\n")
        writeParams(logFile)
    else:
        logFile.write("ERROR: Project not successfully updated. The
following messages were found:\n")
        for msg in GetMessages():
            msgString = "   " + msg.DateTimeStamp.ToString() + " " +
            msg.MessageType + ": " + msg.Summary + "\n"
            logFile.write(msgString + "\n")

    # In any case, save the modified project to the new directory
    projPath = os.path.join(newDir,projFile)
    Save(FilePath=projPath)

    logFile.write("\n")

logFile.close()

Log File  The log file generated by the above script should look like the following:

Processing project in C:\Users\neUser\Demo\ScriptExample1\Original\AsymmetricFlow.wbpj
Parameter values in original project:
P1: PressureDropCoeff = 34.2088
P2: mixing = 0.217835
P5: maxVorticity = 3939.22 [s^-1]
P4: aveVorticity = 27.4697 [s^-1]
Parameter values in revised project:
P1: PressureDropCoeff = 34.0394
P2: mixing = 0.276673
P5: maxVorticity = 3939.22 [s^-1]
P4: aveVorticity = 27.4034 [s^-1]

Processing project in C:\Users\neUser\Demo\ScriptExample1\Original\BaseAnalysis.wbpj
Parameter values in original project:
P1: PressureDropCoeff = 30.04
P2: mixing = 0.248514
P5: maxVorticity = 3939.22 [s^-1]
P4: aveVorticity = 36.8447 [s^-1]
Parameter values in revised project:
P1: PressureDropCoeff = 30.3782
P2: mixing = 0.288321
P5: maxVorticity = 3939.22 [s^-1]
P4: aveVorticity = 36.6682 [s^-1]

Parameter values in original project:
P1: PressureDropCoeff = 29.0038
P2: mixing = 0.266388
P5: maxVorticity = 3939.22 [s^-1]
P4: aveVorticity = 29.2073 [s^-1]

Processing project in C:\Users\neUser\Demo/ScriptExample1/Original/FineAnalysis.wbpj
Parameter values in original project:
P1: PressureDropCoeff = 29.0038
P2: mixing = 0.266388
P5: maxVorticity = 3939.22 [s^-1]
P4: aveVorticity = 29.2073 [s^-1]
Parameter values in revised project:
P1: PressureDropCoeff = 30.7209
P2: mixing = 0.295078
P5: maxVorticity = 3939.22 [s^-1]
P4: aveVorticity = 37.5408 [s^-1]

Discussion  This example demonstrates a number of the typical programming constructs and ANSYS Workbench functionality that will be employed in the creation of scripts. The following discussion refers to the specified line numbers of the example script to illustrate some of these concepts and constructs.

Lines 1-2
The import keyword is used to include Python modules that enhance functionality available within the script. This `os` module used in this script provides capabilities for interacting with operating system services. Refer to the Python Standard Library documentation for details on all available modules.

Lines 4-9
A common pattern is to encapsulate repeated operations in Python function definitions and then call these functions later in the script. Because the script is processed sequentially, the function must be defined before it is used. In this example, we define a function writeParams() to loop through all the parameters in the current project and write information to a provided file object. When concatenating data into a string for output, all non-String data types must first be explicitly converted to their string representation using the `str(<data>)` function or `ToString()` method.

Lines 11-16
We recommend that you use the AbsUserPathName function when working with directory or file paths to promote portability of the path between user accounts and operating systems. See File Path Handling in ANSYS Workbench (p. 14) for further details.

Lines 18-19
Standard Python functionality is used to open and write output to a log file throughout this script.

Lines 21-29
This section also employs standard Python to loop through all file names in a directory. All those file names that end with the .wbpj extension are added to a list of project files for further processing.

Lines 31-38
The script now processes each project in turn. At the beginning of the loop, the full path to the project file is generated from the provided directory name and project file name in a platform-independent manner using Python's os.path.join function. The project is then opened and messages are cleared so that later message processing will focus on new messages generated during the update. Note that the Python is case sensitive. The Open command is part of the ANSYS Workbench Project namespace and is used to open an ANSYS Workbench project, while the open command is provide by Python and is used to open any file and create a Python file object.
The `writeParams` function defined earlier in the script is used to record parameter values to the log file before project modification.

The `GetAllSystems()` query is used to provide references to all systems in the project.

Since we are specifically interested in Geometry components, we use the system's `GetContainer()` query to try to get a reference to the Geometry data. A Python `try/except/else` construct is used to handle the success or failure of the query.

An alternate approach here would be to walk through each component in the system, and then use information about the component to decide which to process. That approach is demonstrated in a subsequent example.

An ANSYS Workbench project update is used to recompute all aspects of the project affected by the geometry change. This step is the most computationally expensive part of the script.

After the update operation has completed, we check if the whole project has been successfully updated. If the project update was successful, we call the `writeParams` function again to output updated parameter information to the log file. If the update was not successful, we record any project messages that were generated during the update.

Similar to the method we used to process parameters, we loop through all messages in the project and write key information about each message to the log file.

A file path for the project within the desired directory for the modified projects is generated, and the ANSYS Workbench Save command is called to save the modified project. After saving, the loop repeats for the next project in the list.

In this example, we will look at how you can interact and query specific files that are associated with components in a project. In this example, you wish to know how and where a specific geometry file has been used within any ANSYS Workbench project that you have within your directory.

To automate this search, we will write a script that performs the operations described below. Two different methods that accomplish the same task are demonstrated to illustrate different approaches to project and file navigation.

1. Starting from a given location, recursively search through all subdirectories to find any ANSYS Workbench projects.

2. For each project, looks for a specified geometry file in using one of two methods:
   a. Get a flat list of all files in the project, and look for the desired filename.
   b. Walk through all systems and their components, and query each component to get the list of files registered to it. Then look if the component is using a file of the specified name. This method is a
more involved but provides more detailed information (e.g., the specific system/component using the file) and gives access to the ANSYS Workbench FileReference to provide detailed information like file size and last modification time.

**Workbench Script**  The script for this example follows. Each line is numbered for reference in the discussion that follows.

```python
1 # import the 'os' module, which provides a portable way of using operating system dependent functionality
2 import os
3
4 # A helper function to find all Workbench projects within a specified directory and add them to a list.
5 # This recursively calls itself to process subdirectories.
6 def findProjectFiles(searchDir, projList):
7     print "Searching in %s" % searchDir
8     for dirEntry in os.listdir(searchDir):
9         fullName = os.path.join(searchDir, dirEntry)
10         if dirEntry.endswith(".wbpj"):
11             # Store the full path to the project file
12             projList.append(fullName)
13         if os.path.isdir(fullName):
14             findProjectFiles(fullName, projList)
15
16 # Define starting directory to find project files.
17 # Here we'll look everywhere within the user's project path root directory.
18 searchDir = GetUserPathRoot()
19
20 # Define file name of interest
21 targetFile = "pipe.x_t"
22
23 # Recursively find all WB2 projects within the search directory
24 projList = []
25 findProjectFiles(searchDir, projList)
26
27 # Open a log file to record script progress
28 logFile = open(AbsUserPathName("FindFileInProjects.log"),"w")
29
30 for projFile in projList:
31     try:
32         Open(FilePath=projFile)
33     except:
34         logFile.write("Error opening %s\n" % projFile)
35         continue
36
37 # Method 1: Search the list of all project files.
38 # This method is simpler, but not as much file information is readily available
39 for fileName in GetAllFiles():
40     if fileName.find(targetFile) > -1:
41         logFile.write("---\n")
42         logFile.write("File %s found in project %s\n" % (fileName, projFile))
43         logFile.flush()
44
45 # Method 2: Walk through the systems and components, to find any that are using the file.
46 # This method is more complex, but provides detailed information through systems,
47 # components and file references. It's also a useful example of general
48 # System & Component navigation.
49
50 # Loop over all systems in the project
51 for system in GetAllSystems():
52     try:
53         for component in system.Components:
54             container = component.DataContainer
55             if compFile.FileName.find(targetFile) > -1:
56                 logFile.write("---\n")
57                 sysStr = "Target file found in component %s of system named %s. Details:\n"
58                 fileStr = "  %s, Size %s bytes, Modified %s\n"
59                 logFile.write(sysStr % (component.DisplayText, system.DisplayText, projFile))
60         logFile.write("---\n")
61         logFile.write(fileStr % (fileName, projFile))
62     except:
63         logFile.write("Error opening %s\n" % projFile)
```

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
LogFile.write(fileStr % (compFile.Location, compFile.Size, compFile.LastModifiedTime))
LogFile.flush()
LogFile.close()

Log File  The log file generated by the above script should look like the following:

--
Target file found in component Geometry of system named Static Structural in C:\Users\neUser\DemoProjects\pipe1.wbpj. Details:
  E:\data\Models\pipe_demo\pipe.x_t, Size 6934 bytes, Modified 06/07/2009 11:50:53 AM
--
File E:\data\Models\pipe_demo\pipe.x_t found in project C:\Users\neUser\DemoProjects\pipe1.wbpj
--
Target file found in component Geometry of system named Static Structural in C:\Users\neUser\Working\pipeDemo.wbpj. Details:
  E:\data\Models\pipe_demo\pipe.x_t, Size 6934 bytes, Modified 06/07/2009 11:50:53 AM
--
File E:\data\Models\pipe_demo\pipe.x_t found in project C:\Users\neUser\Working\pipeDemo.wbpj

Discussion  This example demonstrates how to navigate through systems and components in a project, as well as useful queries and data entities for working with files. The following discussion refers to the specified line numbers of the example script to illustrate some of these concepts and constructs. Discussion points from earlier examples will not be repeated here.

Lines 4-14
This function finds all project files within a specified directory. The Python os.listdir function returns all file and directory names in the specified location. If the name ends with .wbpj, ANSYS Workbench stores the full path to a list of projects. If the name is a directory, ANSYS Workbench recursively calls the function to search the subdirectory.

Lines 32-36
Here, the script demonstrates exception handling to catch any errors that result from opening the project file and report the error to the log file. The continue statement skips to the next entry in the project file loop.

Lines 38-45
This method uses the GetAllFiles() query to get a flat list of all files used in the project and looks at each entry to see if contains the target file name. If so, we record the full path to the target file and the project file to the log.

Line 53
The GetAllSystems() query is used to loop over all systems in the project.

Line 55
The Components property in the system is accessed to loop over the set of components in the system.

Line 56
The DataContainer property is used to access the data within the component.

Line 58
The GetFiles() method on a container is used to return a list of FileReferences that have been associated to that container.

Lines 59-65
We look at the FileName property of each FileReference to see if it contains the target file name. If so, other properties of the FileReference, System and Component are used to record information about the file and its location within the project to the log file.
Material Properties and Tabular Data

This example demonstrates scripting interaction with Engineering Data to access, create, and modify material property data. In this example, you have experimental total strain/stress data in a text file that you wish to use to define Multilinear Isotropic Hardening property data for a material.

As an additional consideration, Multilinear Isotropic Hardening property data in Engineering Data is defined in terms of plastic strain. The script must first convert the total strain data to plastic strain, using the relationship:

\[
\text{Plastic Strain} = \text{Total Strain} - \frac{\text{Stress}}{\text{Young's Modulus}}
\]

The above consideration will be used to demonstrate calculations based on physical quantities and units.

To automate property creation, we will write a script that performs the operations described below:

1. Create a system for a Static Structural analysis, and access the data container for Engineering Data.
2. Load material data for Polyethylene. By default, this material does not include property data for Multilinear Isotropic Hardening.
3. Read experimental data from a text file, and generate lists of data for the necessary variables (converting Total Strain to Plastic Strain)
4. Create the Multilinear Isotropic Hardening property within Polyethylene and set its data table.

Sample Data File  The sample data file to be used in this example follows:

```
# Stress Strain Data for the Material Properties scripting example.
# The data is Total Strain (in m m^-1), Stress (in MPa)
#
7.33E-02, 80.6
1.80E-01, 88.0
6.30E-01, 142.5
7.53E-01, 168.0
8.70E-01, 187.0
```

Workbench Script  The script for this example follows. Each line is numbered for reference in the discussion that follows.

```
# Create a Static Structural system and access its data container for Engineering Data
template1 = GetTemplate(
    TemplateName="Static Structural",
    Solver="ANSYS")
system1 = template1.CreateSystem()
enengineeringData1 = system1.GetContainer(ComponentName="Engineering Data")

# Import Polyethylene
poly = engineeringData1.ImportMaterial(
    Name="Polyethylene",
    Source="General_Materials.xml")

# Initialize lists for variable values
temperature = []
strain = []
stress = []

# Get the value of Young's Modulus for use in the Total/Plastic strain calculation.
# Material Property data is always returned as a Quantity.
elasticity = poly.GetProperty(Name="Elasticity")
```
Discussion  This example demonstrates interaction with Engineering Data to get, create, and set material properties, as well as calculations involving quantities and units. The following discussion refers to the specified line numbers of the example script to illustrate some of these concepts and constructs. Discussion points from earlier examples will not be repeated here.

Lines 1-11
To set the project up to work with Polyethylene, we create a new Static Structural system, access its Engineering Data container, and load material data from the General Materials library.

Lines 13-16
Interaction with tabular data is done in terms of the variables that make up the data table. Each variable represents a column in the table, with a list of data for the variable. Here we initialize three lists for temperature, strain and stress data. Multilinear Isotropic Hardening property data can be temperature-dependent, and temperature is a required variable in the data table. Since our data is not temperature-dependent, we will use the same value of temperature (0 [C]) at each strain/stress point.

Lines 18-21
Since the calculation for plastic strain depends on Young's Modulus, we use the GetData() method to get the Young's Modulus value from the elasticity property. Within Engineering Data, all property values are stored and returned as quantities which include both a numeric value and a unit. See Lines 35-41 for further discussion on quantities and units.

Lines 23-30
Here we open our text data file and read each line one at a time. If the line begins with a comment character '#', we continue to the next line in the file.
Lines 32-33
The Python `split()` function is used to break the comma delimited line and return the separate values into variables for strain and stress.

Lines 35-37
In this example we are creating explicit unit-based quantities for stress and strain values, to ensure dimensional and unit consistency in later calculations.

Material property data (or any other quantity data) can be set either as a quantity (i.e., with value and units), or as a simple numeric value (see Specifying Quantities Without Units). If a numeric value is supplied, the units are assumed to be the same as the current project units for the variable (it is the script writer's responsibility to ensure the numeric data and project units are consistent).

Line 40
Since temperature is a simple constant value, we will specify temperature as a numeric value (without units), and the units will be taken from the current project unit system. We append 0 into the list of temperature data for this strain/stress pair.

Line 41
Here we calculate plastic strain based on the available quantities and append it to the variable list. Mathematical operations involving quantities enforce dimensional consistency and automatically perform unit conversion as necessary to generate a consistent result.

Line 42
Stress data is also appended to the appropriate list.

Lines 45-47
A new property for Multilinear Isotropic Hardening is created within Polyethylene.

Lines 48-52
The `SetData()` method is used to set single value or tabular data for material properties. Some properties can contain more than one data table, so the `SheetName` and `SheetQualifiers` arguments are used to specify the exact data table to be accessed. The `Variables` argument specifies one or more variables in the table to be set. The `Values` argument specifies the values that correspond to each variable.

Lines 54-57
Finally we save the project, overwriting if one already exists.

**Mechanical APDL and Sending Commands to Integrated Applications**

As discussed in Scripting and Data-Integrated Applications (p. 6), ANSYS Workbench can interact with the native scripting language of many of its integrated applications. This example demonstrates scripting interaction with Mechanical APDL and the use of the `SendCommand` method to pass APDL commands to the editor.

The case under consideration is a simple stress analysis of a bar that is defined in a Mechanical APDL input file. The bar dimensions and output displacement have been parameterized within Mechanical APDL. In this example, you wish to write a script that automates this analysis for a number of different bar lengths, and write an ANSYS .cdb file for each case for future processing.

To automate this process, we will write an ANSYS Workbench script that performs the operations described below:

1. Create a system for Mechanical APDL analysis, and loads the specified ANSYS input file.
2. Publishes some of the parameters from the input file to ANSYS Workbench.

3. Loops through a list of desired bar lengths and, for each value, does the following:
   a. Sets the length parameter value.
   b. Updates the project to compute displacement for the new length.
   c. Sends APDL commands to Mechanical APDL to write the .cdb file to a desired location.

Sample Data File  The sample data file to be used in this example follows:

```plaintext
! set parameters
xlen=3
ylen=4
zlen=7

! define model
/prep7
block,,xlen,,ylen,,zlen
et,1,185
mp,ex,1,1e6
mp,prxy,1,0.3
vmesh,all
nsel,s,loc,z,0
da,all,all
nsel,s,loc,y,ylen
d,all,all
sf,all,pres,125
alls
fini

! obtain the solution
/solution
solve
fini

! retrieve results
/post1
! get the max stress at the fixed end
nsel,s,loc,x,xlen
nsel,r,loc,y,ylen
nsel,r,loc,z,0
*GET,out_my_node_stress,NODE,1,NXTH
*GET,out_seqv,NODE,out_my_node_stress,S,EQV
! get the max displacement at the free end
nsel,s,loc,x,xlen
nsel,r,loc,y,ylen
nsel,r,loc,z,zlen
*GET,out_my_node_def,NODE,1,NXTH
*GET,out_uy,NODE,out_my_node_def,U,Y
alls
fini
```

Workbench Script  The script for this example follows. Each line is numbered for reference in the discussion that follows.

```plaintext
1 # Import the 'os' module, which provides a portable way of using
2 # operating system dependent functionality
3 import os
4
5 # Specify the Mechanical APDL Input file to be processed
6 inputfile = AbsUserPathName("Demo/ScriptExample4/bar.dat")
7
8 # Provide a list of bar length (2) values to solve and write CDB files for.
9 # Note: We expect '0' to fail.
10 zValues = [3,5,0,12,15]
11
12 # Open a log file to record script progress
```
13 logFile = open(AbsUserPathName("my_bar_script.log"), "w")
14
15 # Start a new project and create the Mechanical APDL system
16 Reset()
17 template1 = GetTemplate(TemplateName="Mechanical APDL")
18 system1 = template1.CreateSystem()
19
20 # Read the input file into the Mechanical APDL Setup
21 setup1 = system1.GetContainer(ComponentName="Setup")
22 mapdlInputFile1 = setup1.AddInputFile(FilePath=inputFile)
23
24 # Create Workbench parameters from two of the Mechanical APDL parameters
25 # in the input file
26 mapdlInputFile1.PublishMapdlParameter(Name="ZLEN")
27 parameter1 = Parameters.GetParameter(Name="P1")
28
29 mapdlInputFile1.PublishMapdlParameter(
30 Name="OUT_UY",
31 IsDirectOutput=True)
32 parameter2 = Parameters.GetParameter(Name="P2")
33
34 # Save the initial project definition.
35 Save(
36 FilePath=AbsUserPathName("Demo/ScriptExample4/myBar.wbpj"),
37 Overwrite=True)
38
39 # Loop through all provided bar lengths
40 for zVal in zValues:
41
42     # Set the Z (length) parameter expression
43     parameter1.Expression = str(zVal)
44     logFile.write("Updating for z = %s\n" % zVal)
45
46     # Update the project for the new parameter value, and report
47     # success or failure to the log file.
48     try:
49         Update()
50     except:
51         logFile.write(" Update failed.
")
52     else:
53         logFile.write(" Update succeeded. UY = %s\n" % parameter2.Value)
54
55     # Generate the name of the CDB file to save
56     cdbName = os.path.join(GetUserFilesDirectory(), "my_bar_%s\n" % cdbName)
57     cdbNameForCmd = cdbName.replace("\\", "/")
58     if os.path.exists(cdbName):
59         os.remove(cdbName)
60     apdlCmd = "cdwr,db,%s" % cdbNameForCmd
61     setup1.SendCommand(Command=apdlCmd)
62     logFile.write(" CDB written to %s\n" % cdbName)
63
64 # Save the final project state.
65 Save()
66 logFile.close()

Log File  The log file generated by the above script should look like the following:

Change all the forward slashes (\/) with back slashes (/)

Updating for z = 3
Update succeeded. UY = -0.00083352729
CDB written to C:\Users\neUser\Demo\ScriptExample4\myBar_files\user_files\my_bar_3.cdb
Updating for z = 5
Update succeeded. UY = -0.00304172391
CDB written to C:\Users\neUser\Demo\ScriptExample4\myBar_files\user_files\my_bar_5.cdb
Updating for z = 0
Update failed.
CDB written to C:\Users\neUser\Demo\ScriptExample4\myBar_files\user_files\my_bar_0.cdb
Discussion  This example demonstrates interaction with Mechanical APDL to operate on an existing ANSYS input file, and send APDL commands. The following discussion refers to the specified line numbers of the example script to illustrate some of these concepts and constructs. Discussion points from earlier examples will not be repeated here.

Lines 1-13
The initial lines of the script import useful modules, define controlling variables and create a log file to record script progress.

Lines 15-18
Here we start a new project and create a new Mechanical APDL system.

Lines 20-22
We use the AddInputFile method on the Mechanical APDL Setup container to read and process the specified input file.

Lines 24-33
The input file contains a number of values that could be parameterized. Here we promote two of them to be parameters that are controlled and displayed at the ANSYS Workbench level. We access the ZLEN parameter as an input to set the bar length, and the OUT_UY parameter as an output to track maximum Y displacement.

Lines 34-37
We save the project to a permanent location. Until the project is saved, all files and project directories are based on a temporary location. By saving the project, we can more accurately report the directory that holds the .cdb files. The script will still function if the project is not first saved, but will report the temporary location for the project files.

Lines 39-44
We start the loop for the desired bar length values and set the Z parameter appropriately. Note that the Expression for a parameter is a string as it can support complex functions. The Python str() function is used to convert the Z value to a string.

Lines 46-53
We call the project Update() command to recalculate the project based on the new length parameter value, and use Python exception handling to report the success or failure of the update. The value of output displacement is reported on successful update.

Lines 55-57
We use the project user_files directory to store the .cdb files for each case. Here we generate the desired .cdb file path based on the GetUserFilesDirectory() query and use os.path.join to combine that with the desired file name.

Mechanical APDL typically uses forward slashes in file paths by convention, so we convert any backslashes to forward in line 62.

Line 59-62
Mechanical APDL interactively prompts you to overwrite a .cdb file if one already exists, so we ensure no .cdb file of the desired name is present.
We generate an APDL command string to write the .cdb file to the desired file path and use the SendCommand method to execute it. While this example just passes a single line command, the command string can contain multiple commands separated by newlines.

We complete the script by saving the final project state and closing the log file.

**Updating a Workbench Project and Parameters from Excel**

This example demonstrates the mechanics of interacting with Microsoft Excel from within an ANSYS Workbench script, and connecting ANSYS Workbench operations to UI Control events issues by Microsoft Excel (or other similar interfaces). The physics in this example is a simple cantilever beam, where the beam length and load are input parameters, and the resulting deflection is an output parameter. The case is defined within ANSYS Workbench as a Static Structural system with the appropriate parameters created.

Of primary interest in this example is that user interaction with this project will be done via Microsoft Excel, where you will:

1. Set the input parameter values within an Excel Workbook.
2. Press an ‘Update Workbench’ button in Excel.
3. See an ‘Updating…’ message in Excel as the calculation proceeds, and the output parameter value updated within the Workbook when the calculation is complete.

When the ANSYS Workbench script below is executed, it opens the necessary ANSYS Workbench project and Excel Workbook, and then establishes an event connection so that an ANSYS Workbench script function is executed at the press of the button within Excel.

You will need additional files to complete this example. These files are located in `<os drive>\Program Files\Ansys Inc\V150\commonfiles\examples\Scripting` (Windows) or `/an-sys_inc/v150/commonfiles/examples/Scripting` (Linux). You should copy these files to your working directory.

- ParameterExample.xlsx
- ExcelParameterScripting.zip -- extract to your working directory.
Workbench Script  The script for this example follows. Each line is numbered for reference in the discussion that follows.

1 # IronPython imports to enable Excel interop
2 import clr
3 clr.AddReference("Microsoft.Office.Interop.Excel")
4 import Microsoft.Office.Interop.Excel as Excel
5
6 workingDir = AbsUserPathName("Demo/WBScriptAndConnectionExamples/SE5_ExcelIntegration/")
7
8 def updateHandler():
9   
10      # Define key ranges in the Workbook
11      lengthCell = worksheet.Range["A3"]
12      loadCell = worksheet.Range["B3"]
13      defCell = worksheet.Range["C3"]
14
15      # Get the Workbench Parameters
16      lengthParam = Parameters.GetParameter(Name="P1")
17      loadParam = Parameters.GetParameter(Name="P2")
18      defParam = Parameters.GetParameter(Name="P3")
19
20      # Assign values to the input parameters
21      lengthParam.Expression = lengthCell.Value2.ToString()
23
24      # Mark the deformation parameter as updating in the workbook
25      defCell.Value2="Updating..."
26
27      # Run the project update
28      Update()
29
30      # Update the workbook value from the WB parameter
31      defCell.Value2 = defParam.Value.Value
32
33
34 # Open the Workbench Project
35 Open(FilePath = workingDir + "ExcelParameterScripting.wbpj")
36
37 # Open Excel and the workbook
38 ex = Excel.ApplicationClass()
39 ex.Visible = True
40 workbook = ex.Workbooks.Open(workingDir + "ParameterExample.xlsx")
41 worksheet = workbook.ActiveSheet
42
43 #Apply the update handler to the workbook button
44 OLEbutton = worksheet.OLEObjects("CommandButton1")
45 OLEbutton.Object.CLICK += updateHandler

Discussion  This example demonstrates a number of the typical programming constructs necessary for ANSYS Workbench scripting to interact with the Common Language Runtime (CLR) API exposed by Microsoft Excel and similar applications.

Lines 1-4
The clr module is imported to enable IronPython to load and interact with CLR modules from other applications. In this example we add a CLR reference to the Microsoft Excel Interop assembly, and then import the module as the Excel namespace.

Line 6
The working directory is set, so that the script can load the Project and Excel Workbook from the specified location. When completing this example, this line will need to be modified to reflect your working location.

Lines 8-31
An updateHandler() function is created that performs all the necessary actions to interact with Excel and update the ANSYS Workbench project. This function is connected to the CLICK event of the Excel button on line 45 of this script.
Lines 10-13
Specific cells of interest in the Workbook are created as named references in the script to facilitate later use of these cells. In this instance, the cells that hold the values of the two input parameters and one output parameter are given named references.

Lines 15-18
References to the three parameters exposed by the Static Structural system are also assigned to variables for later use.

Lines 20-22
The Expression property defining the input parameters are set based on the values of the associated cells in the Excel Workbook. Note the ‘Value2’ property is used because it has simpler interaction when working with a single cell value.

Lines 24-25
The script sets the value of the output parameter cell in the workbook to ‘Updating...’ while calculation proceeds.

Lines 27-28
The Update() command is executed to update the project based on the new input parameter values.

Lines 30-31
When the Update is complete, we update the output cell value in the Workbook with the parameter value in the Project.

Note the reference to defparam.Value.Value. The Value property of a parameter can have different types (Numeric, String, Boolean, Quantity, etc). In this instance, it is a Quantity, which in turn has properties for its Value and Unit. Therefore defparam.Value.Value is the numeric part of the Quantity that is the parameter’s current value.

Lines 34-35
The lines following the UpdateHandler definition are those first executed when the script executes. The first operation is to load the Workbench Project containing the parameterized Static Structural analysis.

Lines 37-39
Using the imported Excel namespace, we create an instance of the Excel application and make it visible.

Lines 40-41
We open the Excel workbook that contains the parameter table and ‘Update’ button, and get a reference to the primary (active) worksheet in the book.

Lines 43-44
We get a reference to the named OLE button (CommandButton1) that is present in the worksheet. This button was added to the worksheet by inserting the control within Excel, but no other macros or code associated with this button is required in the workbook.

Line 45
The updateHandler() function is added as an event handler to the CLICK event on the command button. Whenever the button is clicked, the associated ANSYS Workbench script function is executed.
Known Issues and Limitations

Known Issues and Limitations

A listing of known Journaling and Scripting issues and limitations.

**Parameters**

- The behavior for Parameters.GetAllRetainedDesignPoints has changed at Release 15.0. The query no longer returns the collection of all exported design points. Instead it is based on the retained design point concept which is beta functionality at 15. Users should replace their script calls with Parameters.GetAllExportedDesignPoints to maintain original behavior. Please review the Workbench Help for more details regarding retained and exported design points.
Part I: Data Containers
Ansoft

This container holds data for an instance of an Ansoft analysis.

Data Entities

AnsoftSolutionEntityType

The main entity for the Solution component of Ansoft systems. This entity exposes parameters that control how the update operation is carried out by the underlying editor.

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string  
Read Only: No

KeepDesktopEnginesDuringUpdateMultiDPs

Controls whether the various engine processes that the editor employs in its solve process are kept alive across design points when workbench is performing a multiple design point update. This controls the memory/time tradeoff during multiple design point updates.

Type: bool  
Read Only: No

NonGraphical

Control whether the editor is launched in non-graphical mode or with full UI.

Type: bool  
Read Only: No
Ansoft Feedback Iterator

This container holds data related to a two-way coupled system where Maxwell, HFSS, or Q3D is the main upstream system.

Methods

**ExecuteOneTwoWayIteration**

This command executes one two-way iteration among the coupled clients of the master "Two Way iterator" component

**ResetCompletedIterations**

This command rests the number of completed iterations

Data Entities

**FeedbackIteratorEntity**

The main FeedbackIterator entity holding information about the coupled clients etc.

Properties

**CallbackPath**

The callback script file

Type: string

Read Only: No

**CompletedSolveIterations**

The number of completed solve iterations

Type: int

Read Only: No

**CurrentDeltaDPercentage**

The current delta D % we have on record.

Type: string
Read Only  Yes

**CurrentDeltaTPercentage**

The current delta T % we have on record.

**Type**  string

Read Only  Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  string

Read Only  No

**MaxSolverIterations**

The Max number of solve iterations allowed when the FBIter is working with a system that sends convergence information

**Type**  int

Read Only  No

**TargetDeltaDPercentage**

The relative tolerance value that indicates convergence for displacement

**Type**  double

Read Only  No

**TargetDeltaTPercentage**

The relative tolerance value that indicates convergence for temperature

**Type**  double

Read Only  No

**TotalSolverIterations**

The total number of solve iterations requested by the user.

**Type**  int

Read Only  No

**FeedbackIteratorTransferManagerEntity**

The entity that handles the transfer data protocol for the AnsoftCADGeometry Addin container.
Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string

Read Only: No
AQWA

AQWA Model
This container holds Model data for an instance of AQWA.

Methods

Edit
Opens the AQWA editor to allow modification of AQWA Model data.

Exit
Exits the AQWA editor.

GetModel
Get the DataReference for the Model data entity.

Return
DataReference for the Model data entity.

Type
DataReference

SendCommand
Executes one or more JScript commands in the AQWA editor.

Required Arguments

Command
The command to execute in the AQWA editor.

Type
string

Example
To execute some arbitrary command (in this case, causing a dialog box to appear) in the AQWA editor:

model1.SendCommand(Command="WBScript.Out(\"My Text\",true);\n}"

If the AQWA editor is not open SendCommand will open it, run the command and then close it. Consider calling model1.Edit() to open the editor before using SendCommand if you do not want it to close.
Data Entities

**AqwaModel**

Model data entity.

**Properties**

**AnalysisType**

Entity Analysis Type setting.

- **Type**: string
- **Read Only**: Yes

**AQDBDatabaseFilesWritten**

Indicates if aqwa .aqdb database files have been written for this system.

- **Type**: bool
- **Read Only**: No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**GeometrySelected**

Specifies if this data entity has Geometry data available.

- **Type**: bool
- **Read Only**: No

**PhysicsType**

Entity Physics Type setting.

- **Type**: string
- **Read Only**: Yes

**ProjectName**

The system string identifier.

- **Type**: string
- **Read Only**: No
**SolverType**

Entity Solver Type setting.

**Type** string

**Read Only** Yes

---

**AQWA Results**

This container holds Results data for an instance of AQWA.

**Methods**

**Edit**

Opens the AQWA editor to allow modification of AQWA Setup data or viewing of the AQWA Results.

**Exit**

Exits the AQWA editor.

**GetResults**

Get the DataReference for the Results data entity.

**Return** DataReference for the Results data entity.

**Type** DataReference

**SendCommand**

Executes one or more JScript commands in the AQWA editor.

**Required Arguments**

**Command**

The command to execute in the AQWA editor.

**Type** string

**Example**

To execute some arbitrary command (in this case, causing a dialog box to appear) in the AQWA editor:

```jscript
modell.SendCommand(Command="WBScript.Out("My Text",true);" )
```

If the AQWA editor is not open SendCommand will open it, run the command and then close it. Consider calling model1.Edit() to open the editor before using SendCommand if you do not want it to close.
Data Entities

*AqwaResults*

Results data entity.

**Properties**

*AnalysisType*

Entity Analysis Type setting.

**Type** string

Read Only Yes

*DisplayText*

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

Read Only No

*PhysicsType*

Entity Physics Type setting.

**Type** string

Read Only Yes

*SolverType*

Entity Solver Type setting.

**Type** string

Read Only Yes

---

**AQWA Setup**

This container holds Setup data for an instance of AQWA.

**Methods**

*Edit*

Opens the AQWA editor to allow modification of AQWA Setup data or viewing of the AQWA Results.
**Exit**

Exits the AQWA editor.

**GetSetup**

Get the DataReference for the Setup data entity.

**Return**

DataReference for the Setup data entity.

**Type**

DataReference

**SendCommand**

Executes one or more JScript commands in the AQWA editor.

**Required Arguments**

**Command**

The command to execute in the AQWA editor.

**Type**

string

**Example**

To execute some arbitrary command (in this case, causing a dialog box to appear) in the AQWA editor:

```javascript
model1.SendCommand(Command="WBScript.Out("My Text",true);" )
```

If the AQWA editor is not open SendCommand will open it, run the command and then close it. Consider calling model1.Edit() to open the editor before using SendCommand if you do not want it to close.

**Data Entities**

**AqwaSetup**

Setup data entity.

**Properties**

**AnalysisType**

Entity Analysis Type setting.

**Type**

string

**Read Only**

Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**

string
Read Only No

**PhysicsType**

Entity Physics Type setting.

Type `string`

Read Only Yes

**SolverType**

Entity Solver Type setting.

Type `string`

Read Only Yes

**StaticOnly**

Used to control if Hydrodynamic Diffraction analysis solve is full or hydrostatic only when run from Workbench. If True then only Hydrostatics are solved.

Type `bool`

Read Only No

---

**AQWA Solution**

This container holds Solution data for an instance of AQWA.

**Methods**

**Edit**

Opens the AQWA editor to allow modification of AQWA Solution data.

**Exit**

Exits the AQWA editor.

**GetSolution**

Get the DataReference for the Solution data entity.

**Return**

DataReference for the Solution data entity.

Type `DataReference`

**SendCommand**

Executes one or more JScript commands in the AQWA editor.
Required Arguments

**Command**
The command to execute in the AQWA editor.

**Type** string

**Example**
To execute some arbitrary command (in this case, causing a dialog box to appear) in the AQWA editor:

```plaintext
model1.SendCommand(Command="WBScript.Out("My Text",true);"
```

If the AQWA editor is not open SendCommand will open it, run the command and then close it. Consider calling model1>Edit()to open the editor before using SendCommand if you do not want it to close.

Data Entities

**AqwaSolution**
Solution data entity.

**Properties**

**AnalysisType**
Entity Analysis Type setting.

**Type** string

**Read Only** Yes

**DisplayText**
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

**Read Only** No

**PhysicsType**
Entity Physics Type setting.

**Type** string

**Read Only** Yes

**SolverType**
Entity Solver Type setting.

**Type** string
Read Only  Yes
AUTODYN

AUTODYN Analysis
This container holds Results data for an instance of AUTODYN.

Methods

GetAutodynAnalysis
Returns a reference to an AutodynAnalysis data entity within the container.

Return A reference to the AutodynAnalysis data entity.

Type DataReference

AUTODYN Setup
This container holds Solution data for an instance of AUTODYN.

Methods

Edit
Opens the AUTODYN editor for pre-processing, solving and post-processing. If there is an input file associated with the system the editor will load that file when it opens, otherwise the editor will create a new model.

Exit
Closes the AUTODYN editor. If there is any unsaved data in the editor it will be saved.

GetAutodynSetup
Returns a reference to an AutodynSetup data entity within the container.

Return A reference to the AutodynSetup data entity.

Type DataReference

Import
Specifies the AUTODYN input file (*.ad), to be associated with the system. The specified file is copied to the systems working directory and registered with workbench.
Required Arguments

**FilePath**
The location of the AUTODYN input file (*.ad). If the name is blank any currently associated input file will be removed.

*Type*  string

**SetUserExecutable**
Sets the location of the user created executable to be used when the editor is opened.

Required Arguments

**ExecutablePath**
The full path and name of the executable to use when pre-processing, solving and post-processing.

*Type*  string

Data Entities

**AutodynSetup**
This holds the properties of the setup component.

Properties

**Directory**
The working directory for the editor.

*Type*  string

*Read Only*  No

**DisplayText**
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type*  string

*Read Only*  No

**FileName**
The name of the associated input file.

*Type*  string

*Read Only*  No

**strUserExecutable**
The location of the user defined executable to use.
<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>Yes</td>
</tr>
</tbody>
</table>
CFD Results

This container holds Results and post-processing data for a CFD simulation.

Methods

**Edit**

Opens the CFD-Post editor to allow modification of Results data.

This command will open the editor only if one is not already open on this component. If this component's editor is already open and in interactive mode, then it will be raised to the front.

**Optional Arguments**

- **Interactive**
  - Run the editor in interactive mode if True, or in no GUI mode if False.
  - If not specified, the editor runs in interactive mode.
  - **Type** bool
  - **Default Value** True

**Exit**

Exits the editor.

Any changes made in this editor will be retained on exit. These changes are made permanent by a Project Save, and will be discarded in the event of closing the project without saving.

If no editor is open on the component in question, this command will have no effect.

**GetCFDResults**

Returns the Data Entity which contains user settings and properties for the Results container.

- **Return** The Data Entity containing settings for this component.
  - **Type** DataReference

**SendCommand**

Sends commands to the editor for this component using CFX Command Language (CCL) syntax. If the editor for this component is not open, it will be launched before the commands are sent and sub-
sequently closed. In this mode, component data is loaded and saved as if calling Edit(Interactive=False) and Exit around the SendCommand invocation.

The instructions must be CFX Command Language session commands that are valid for the editor in question.

**Required Arguments**

<table>
<thead>
<tr>
<th>Command</th>
<th>Valid CFX Command Language (CCL) commands</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>string</td>
</tr>
</tbody>
</table>

**Data Entities**

**CFDResults**

Entity which manages settings and data for the CFD Results component.

**Properties**

**ClearState**

Specifies whether the state should be cleared when the Results cell is updated. This is only used if a report has been selected or a session script has been specified to run on update.

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>

**CustomReportTemplate**

Specifies the file path of the custom report template to load if LoadReport is set to 'Custom'. It is recommended that the template file be located in the user_files directory of the project so that it is available if the project is archived and moved to another system.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>

**GenerateReport**

Specifies whether to regenerate a report during component update.

The report will be generated according to the report definition within CFD-Post. Its name and location can be controlled by the ReportName and ReportLocationDirectory advanced user properties on this entity.
**Type** 
bool

**Read Only**
No

**LoadReport**

Report to load when Results cell is edited or updated.

**Type**  
PostReportNamesType

**Read Only**
No

**MRESLoadOptionsMsg**

This property has no effect and is simply used for presentation purposes.

**Type**
string

**Read Only**
No

**PostStateFile**

CFD-Post State file location.

Do not modify this property directly.

**Type**
DataReference

**Read Only**
No

**ReportLocationDirectory**

Specifies a custom directory in which to place the report files.

This is an advanced user property to be used in conjunction with the GenerateReport and ReportName properties. By default this property is empty, in which case the project's user_files directory is used. If a directory path is specified, the directory must exist at the time of component update.

**Type**
string

**Read Only**
No

**ReportName**

Specifies the name of the report to be generated during component update. By default the report name is "Report".

This is an advanced user property to be used in conjunction with the GenerateReport and ReportLocationDirectory properties.

**Type**
string

**Read Only**
No
**RunSessionScript**

Specifies whether a CFD-Post session script will be run as part of the component update.

If enabled, the UpdateScriptCCL property on this entity should be set to contain the session script text.

**Type**  bool

**StateInitializationFile**

Contains CFX Command Language script for CFD-Post, to be executed as part of the component update. This can be used to toggle regeneration of exported data, animation files, etc. In this scenario, the script will only need to be run when loading the results into Post at a time when the Results cell is in the ‘Unfulfilled’ state (i.e. has never been opened yet).

This is an advanced user feature, and caution should be exercised. CFD-Post session files are capable of performing file activity and changing state in ways that are not supported in Workbench sessions. Limit your session script to activities which will not corrupt or circumvent Workbench project data management.

**Type**  string

**UpdateScriptCCL**

Contains CFX Command Language script for CFD-Post, to be executed as part of the component update. This can be used to toggle regeneration of exported data, animation files, etc.

This is an advanced user feature, and caution should be exercised. CFD-Post session files are capable of performing file activity and changing state in ways that are not supported in Workbench sessions. Limit your session script to activities which will not corrupt or circumvent Workbench project data management.

**Type**  string
CFX Setup

This container holds Setup data for an instance of CFX-Pre.

Methods

**Edit**

Opens the CFX-Pre editor to allow modification of CFX Setup data.

This command will open the editor only if one is not already open on this component. If this component’s editor is already open and in interactive mode, then it will be raised to the front.

Optional Arguments

**Interactive**

Run the editor in interactive mode if True, or in no GUI mode if False.

If not specified, the editor runs in interactive mode.

**Type** bool

**Default Value** True

**Exit**

Exits the editor.

Any changes made in this editor will be retained on exit. These changes are made permanent by a Project Save, and will be discarded in the event of closing the project without saving.

If no editor is open on the component in question, this command will have no effect.

**GetCFXSetupProperties**

Returns the Data Entity which contains user settings and properties for the Setup container.

**Return** The Data Entity containing settings for this component.

**Type** DataReference

**Import**

Imports CFX Setup data into the CFX-Pre editor from an existing CFX Case file.
The Case file's contents will be imported into a new case file within the project, managed by the Setup component. This operation is not valid if the component already contains Setup data.

**Required Arguments**

**FilePath**
Path of CFX Case File to be imported.

  **Type** string

**SendCommand**

Sends commands to the editor for this component using CFX Command Language (CCL) syntax. If the editor for this component is not open, it will be launched before the commands are sent and subsequently closed. In this mode, component data is loaded and saved as if calling Edit(Interactive=False) and Exit around the SendCommand invocation.

The instructions must be CFX Command Language session commands that are valid for the editor in question.

**Required Arguments**

**Command**
Valid CFX Command Language (CCL) commands

  **Type** string

**SuppressPhysicsErrors**

Suppresses physics validation errors for the current CFX Setup component.

Normally, a CFX Setup component will not permit an update to proceed while validation errors exist in the case. However, in certain special situations, it may be that particular validation errors are tolerable and the case should be solvable.

This command can be used on CFX Setup components to bring the component from Attention Required to Up-to-Date, allowing the downstream Solution component to be updated.

**Data Entities**

**CFXSetupProperties**

Entity which manages settings and data for the CFX Setup component.

**Properties**

**AllowAssemblyMeshImport**

This is a beta option. Allow the user to import the assembly meshes

  **Type** bool

  **Read Only** No

**CaseFile**

CFX-Pre Case file location.
Do not modify this property directly.

**Type**  
DataReference

**Read Only**  
No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

**PhysicsStatus**

Physics state of the CFX-Pre editor, specifying validation errors, warnings and information on the current case.

This property is updated by CFX-Pre only.

**Type**  
string

**Read Only**  
No

**SolverInputExtendedFiles**

Extended CFX Solver Input Files for multiconfig cases

Do not modify this property directly.

**Type**  
DataReferenceSet

**Read Only**  
No

**SolverInputFile**

CFX-Solver Input file location.

Do not modify this property directly.

**Type**  
DataReference

**Read Only**  
No
**CFX Solution**

This container holds data for a CFX Solution.

**Methods**

**ClearOldSolutionData**

Every time the solver is run for a Solution container, a new set of solution files are generated and the old ones are kept around. This command is used to clear all solution data files from past solver runs except for the last one.

**Required Arguments**

- **KeepReferenced**  
  If true, all old solution data will be removed. Otherwise, only data not referenced in the latest solution will be removed.

  **Type**  
  bool

**DisplayMonitors**

Displays run history and monitors for the most recent Solution update.

CFX-Solver Manager is used for displaying monitors, and (if necessary) will be launched by executing this command.

**Edit**

Open the CFX-Solver Manager editor to allow modification of the run definition for the CFX Solution component.

Note that CFX-SolverManager is a GUI-only component that is not scriptable. This command will not be journaled when invoked through the GUI. For batch operations, omit this command and use SetExecutionControl. If execution control is specified in the Setup component, it may be necessary to use this command with the appropriate value for the ExecutionControlOption parameter. Alternatively, the ExecutionControlOption property of the Solution component can be modified directly in the script to ensure no execution control conflicts are detected.

**Optional Arguments**

- **ExecutionControlOption**  
  Specifies how to handle conflicts in execution control if they exist.

  **Default**  
  use On Execution Control Conflict property setting

  **UseSetupExecutionControl**  
  Edit Run Definition using execution control from the Setup container

  **UseSetupExecutionControlAlways**  
  change On Execution Control Conflict property to 'Use Setup Cell Execution Control' and proceed with Edit Run Definition

---

Release 15.0 - © SAS IP Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
<table>
<thead>
<tr>
<th><strong>UseSolutionExecutionControl</strong></th>
<th><strong>UseSolutionExecutionControlAlways</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit Run Definition using execution control from the Solution container</td>
<td>change On Execution Control Conflict property to 'Use Solution Cell Execution Control' and proceed with Edit Run Definition</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td><strong>ExecutionControlConflictOptions</strong></td>
</tr>
<tr>
<td><strong>Default Value</strong></td>
<td>Default</td>
</tr>
</tbody>
</table>

**EditDefFileInCommandEditor**

Opens the CFX Command Editor to allow modification of the physics in the specified CFX Results or Solver Input file.

This command will simply launch the CCL Command Editor with the provided file path. No further Workbench interaction with the CCL Command Editor is possible, and no further activity of this component is journaled. Do not use this command as part of any batch workflow.

**Required Arguments**

- **FilePath**
  - Path of CFX Solver Input File, or Results File, to be edited
  - **Type**: string

**Exit**

Exits the editor.

Any changes made in this editor will be retained on exit. These changes are made permanent by a Project Save, and will be discarded in the event of closing the project without saving.

If no editor is open on the component in question, this command will have no effect.

**GetCFXSolutionProperties**

Returns the Data Entity which contains user settings and properties for the Solution container.

**Return**

- The Data Entity containing settings for this component.
  - **Type**: DataReference

**GetComponentSettingsForRsmDpUpdate**

This query is used to obtain the ComponentSettingsForRsmDpUpdate object for Journaling and Scripting

**GetSolutionSettings**

This query is used to obtain the solution settings object for Journaling and Scripting
**Import**

Imports CFX Solution data from an existing CFX Results file.

The Results file, as well as all files associated with it (such as transient or multi-configuration files, if they exist), will be copied into the project as the generated data for the Solution component.

This command is intended for use when no Setup has been defined. If no data has been provided to an upstream Setup, or to any upstream Geometry or Mesh components (if they exist), then all of these components will automatically be deleted from the system as a result of this command.

**Required Arguments**

FilePath Path of CFX-Solver Results file to be imported.

  Type string

**MarkUpToDate**

Accept an interrupted Solution as Up-to-Date.

The specified Solution component should be in Interrupted state. As a result of this command, the Solution will be marked Up-to-Date.

**SetExecutionControl**

Sets the CFX-Solver settings (in the form of Execution Control CCL) for a Solution component.

These settings will form the basis of a run definition for the Solution component. However, certain settings, such as the Solver Input File, are always overridden during Update. Other settings, such as all Initialization settings, can be overridden by the Solution component's Initialization Option property as well by as the presence of initialization data provided by other components on the project schematic.

The CCL argument must specify valid Execution Control CCL.

**Required Arguments**

CCL CFX Command Language defining Execution Control for the CFX-Solver.

  Type string

**SwitchToBackgroundMode**

Switch the Update in progress into background mode. This will enable operations that are not allowed during an Update in foreground mode (e.g. Project Save).

This command is not normally useful in a script. Journals may record the invocation of this command after an Update invoke, as the result of GUI activity while the Update is in progress. However, replay of these journals will always wait for the Update invoke to complete before invoking the next command, rendering this step ineffectual.
Data Entities

**CFXSolutionProperties**

Entity which manages settings and data for the CFX Solution component.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

**Read Only** No

**ExecutionControlOption**

Specifies how to handle conflicts in execution control between this entity and the upstream Setup cell. It is only used if execution control has been saved in this entity and the execution control in the DEF file, provided by the upstream Setup cell, has changed since the last run.

- **IssueWarning** In the case of a conflict, a message is displayed and the process is aborted.
- **UseExecutionControlFromSetup** In the case of a conflict, the execution control provided in the DEF file is used.
- **UseExecutionControlFromSolution** In the case of a conflict, the execution control saved in the Solution container is used.

**Type** ExecutionControlSource

**Read Only** No

**InitializationOption**

Specifies the way a Solution will be initialized during Update.

Available options:

- **CurrentSolutionData** Initialize the solution from the current Solution component data (if the data is present and is appropriate).
- **InitialConditions** Always initialize the solution from defined initial conditions.

In cases where no existing solution data is present, this property has no effect.

The default behavior is "CurrentSolutionData"; however this default can be changed in Options->CFX->Default Initialization Option

**Type** InitializationOption

**Read Only** No
**LoadMResOptions**

Specifies how multi-configuration solution data should be treated by a Results component when opened in CFD-Post.

This option has no effect if the Solution component does not contain a results set with multiple configurations.

Available options:

- **AllConfigsSingleCase**: All configurations for this set of results should be loaded and treated as a single case.
- **AllConfigsSeparateCases**: All configurations for this set of results should be loaded, but each configuration should be treated as a separate case.
- **LastConfigOnly**: Only the last configuration should be loaded.

The default behavior is "LastConfigOnly".

**Type**  
**MResOptions**

**Read Only**  
No

**ResultsFile**

Current CFX-Solver Results File location.

**Type**  
**DataReference**

**Read Only**  
Yes

**SolverArguments**

Additional Solver Arguments

**Type**  
**string**

**Read Only**  
No

**SolverPath**

Path to a custom Solver Executable

**Type**  
**string**

**Read Only**  
No
Design Exploration

DX Direct Optimization

This container holds data for an instance of Parameter Direct Optimization.

Methods

**GetModel**

Get the DataReference of the Model. An exception is thrown if the entity is not found.

**Return**

The DataReference of the Model.

**Type**

DataReference

Example

The following example shows how the user can get a Model to change one of its properties.

```csharp
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
dOEModel1.DOETYPE = "eDOETYPE_OSFD"
```

Data Entities

**AmoOptimization**

Entity which performs and manages the DesignXplorer Optimization Method Component

Properties

**Converged**

Convergence state

**Type**

bool

**Read Only**

Yes

**CrossoverProbability**

Maximum Allowable Pareto Percentage

**Type**

double

**Read Only**

No
**CurrentParetoPercentage**

Pareto Percentage

**Type** double

**Read Only** Yes

**CurrentStabilityPercentage**

Stability Percentage

**Type** double

**Read Only** Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

**Read Only** No

**MaximumNumberOfPermutations**

Maximum number of permutations for discrete / manufacturable values.

**Type** int

**Read Only** Yes

**MaxNumCandidates**

Maximum Number of Candidates

**Type** int

**Read Only** No

**MaxNumCycles**

Maximum Number Of Cycles for OSFD algorithm.

**Type** int

**Read Only** No

**MaxNumIterations**

Maximum Number of Iterations

**Type** int

**Read Only** No
**MutationProbability**

Maximum Allowable Pareto Percentage

Type: double

Read Only: No

**NumberOfEvaluations**

Number of Evaluations

Type: int

Read Only: Yes

**NumberOfFailures**

Number of Failures

Type: int

Read Only: Yes

**NumberOfInitialSamples**

Number of Initial Samples

Type: int

Read Only: No

**NumCandidates**

Number of Candidates

Type: int

Read Only: Yes

**NumIterations**

Current Iteration

Type: int

Read Only: Yes

**NumSamplesPerIter**

Number of Samples Per Iteration

Type: int

Read Only: No
**ParetoPercentage**

Maximum Allowable Pareto Percentage

**Type**  
double

**Read Only**  
No

**RandomGeneratorSeed**

LHS Seed

**Type**  
int

**Read Only**  
No

**SampleSetSize**

Size of Generated Sample Set

**Type**  
int

**Read Only**  
Yes

**StabilityCriterion**

Maximum Allowable Pareto Percentage

**Type**  
double

**Read Only**  
No

**TypeOfInitialSampling**

Type Of Initial Sampling

**Type**  
TypeOfInitialSampling

**Read Only**  
No

**AsoOptimization**

Entity which performs and manages the DesignXplorer Optimization Method Component

**Properties**

**Converged**

Convergence state

**Type**  
bool

**Read Only**  
Yes
**ConvergenceTolerance**

Convergence Tolerance

**Type** double

**Read Only** No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

**Read Only** No

**MaximumNumberOfPermutations**

Maximum number of permutations for discrete / manufacturable values.

**Type** int

**Read Only** Yes

**MaxNumberReductionPerIteration**

Maximum Number of Domain Reduction per Iteration

**Type** int

**Read Only** No

**MaxNumCandidates**

Maximum Number of Candidates

**Type** int

**Read Only** No

**MaxNumCycles**

Maximum Number Of Cycles for OSFD algorithm.

**Type** int

**Read Only** No

**MaxNumDomainReductions**

Maximum Number of Domain Reductions

**Type** int

**Read Only** No
MaxNumEvaluations
Maximum Number of Evaluations
Type int
Read Only No

NumberOfEvaluations
Number of Evaluations
Type int
Read Only Yes

NumberOfFailures
Number of Failures
Type int
Read Only Yes

NumberOfInitialSamples
Number of LHS Initial Samples.
Type int
Read Only No

NumberOfScreeningSamples
Number of Screening Samples for the Adaptive Single-Objective optimization method.
Type int
Read Only No

NumberOfStartingPoints
Number of Starting Points for the Adaptive Single-Objective optimization method.
Type int
Read Only No

NumCandidates
Number of Candidates
Type int
Read Only Yes
**PercentageOfDomainReductions**

Percentage of Domain Reductions

**Type**
- **double**

**Read Only**
- No

**RandomGeneratorSeed**

LHS Seed

**Type**
- **int**

**Read Only**
- No

**SampleSetSize**

Size of Generated Sample Set

**Type**
- **int**

**Read Only**
- Yes

**DiscreteLevel**

The data entity which describes a Discrete Level of an Input Parameter.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**
- **string**

**Read Only**
- No

**Index**

Zero-based Index of the discrete level in the list of levels of the owning parameter.

**Type**
- **int**

**Read Only**
- No

**Value**

Value of the DiscreteLevel.

**Type**
- **Object**

**Read Only**
- No
Methods

SetValue

Sets the value of a discrete level entity. A discrete level can have an integer value (e.g. a number if holes, a number of turns, etc) or a string value (e.g. a material name or a geometry file name).

Required Arguments

Value  Value set to the discrete level entity.

Type  Object

Example

The following example shows how to retrieve a discrete level from an input parameter and then change its value.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DiscreteInputParameter = model.GetParameter(Name="P2")
level1 = DiscreteInputParameter.GetDiscreteLevel(Name="Level 1")
level1.SetValue( Value="2500" )
```

InputParameter

The data entity which describes an Input Parameter in DesignXplorer.

Properties

Attribute1

First editable attribute of the distribution for an uncertainty parameter. The nature of the attribute depends on the distribution type. For instance, the first attribute of a Normal distribution is the Mean value.

Type  double

Read Only  No

Attribute2

Second editable attribute of the distribution for an uncertainty parameter. The nature of the attribute depends on the distribution type. For instance, the second attribute of a Normal distribution is the Standard Deviation value. Some distribution type do not have a second attribute.

Type  double

Read Only  No

ConstantValue

Constant value of the Parameter when it is disabled.

Type  Object
Read Only No

**DiscreteLevels**

List of the discrete levels.

**Type** List<DataReference>

Read Only Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

Read Only No

**DistributionLowerBound**

Distribution lower bound of the variation range for an uncertainty Parameter.

**Type** double

Read Only No

**DistributionType**

Distribution type for an uncertainty parameter.

**Type** DistType

Read Only No

**DistributionUpperBound**

Distribution upper bound of the variation range for an uncertainty Parameter.

**Type** double

Read Only No

**Enabled**

True if the Parameter is enabled for the current study.

**Type** bool

Read Only No

**Kurtosis**

Kurtosis value of the distribution for an uncertainty parameter.

**Type** double
**Read Only** Yes

**LowerBound**
Lower bound of the variation range for a Continuous Parameter.

Type double

Read Only No

**Mean**
Mean value of the distribution for an uncertainty parameter.

Type double

Read Only Yes

**Nature**
Nature of the Parameter.

Type ParameterNature

Read Only No

**NumberOfLevels**
Number of levels if the parameter nature is Discrete, or the parameter nature is Continuous and the UseManufacturableValues property is set to True.

Type int

Read Only Yes

**Skewness**
Skewness value of the distribution for an uncertainty parameter.

Type double

Read Only Yes

**StandardDeviation**
Standard deviation value of the distribution for an uncertainty parameter.

Type double

Read Only Yes

**Type**
Type of the Parameter, either a DesignVariable in a GDO context, or an UncertaintyVariable in a SixSigma Analysis context.
**Type** SimulationType

**Read Only** Yes

**Units**

Units

**Type** string

**Read Only** Yes

**UpperBound**

Upper bound of the variation range for a Continuous Parameter.

**Type** double

**Read Only** No

**UseManufacturableValues**

True to restrict the variation of the parameter to defined Manufacturable Values.

**Type** bool

**Read Only** No

**Methods**

**AddDiscreteLevel**

Adds a Discrete Level entity on a discrete input parameter. A discrete level can have an integer value (e.g. a number of holes, a number of turns, etc) or a string value (e.g. a material name or a geometry file name). The command has optional arguments to specify the Name of the level and its Index in the list of levels of the parameter. By default, the new level is added to the end of the list. The various discrete levels of an input parameter represent independent configurations of the project, processed in the order of their creation.

**Return** The created entity.

**Type** DataReference

**Required Arguments**

**Value** The value of the discrete level. Value can be an integer or a string.

**Type** Object

**Optional Arguments**

**DisplayText** DisplayText of the created entity. If not specified, a default name of the form "Level [#]" is used.

**Type** string
Index

The position of the new level in the list of discrete levels of the parameter. Index is zero-based. If it is not specified, the new level is appended to the list.

**Type**    int

**Default Value**    -1

**Example**
The following example shows how to add new discrete levels on a discrete input parameter. The third level is inserted between the two others.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DiscreteInputParameter = model.GetParameter(Name="P2")
DiscreteInputParameter.AddDiscreteLevel(Value=3000, DisplayText="Three thousand turns")
DiscreteInputParameter.AddDiscreteLevel(Value=2000, DisplayText="Two thousand turns")
DiscreteInputParameter.AddDiscreteLevel(Value=5000, Index="1")
```

AddLevels

Adds a list of levels to a continuous input parameter. Each level is a quantity or a real number corresponding to a manufacturable value. The list of levels forms a restriction filter used when post-processing the input parameter. If levels are added outside of the variation range, the lower and upper bounds are adjusted accordingly.

**Required Arguments**

- **Levels**
  
  List of added levels.

  **Type**    List<string>

**Optional Arguments**

- **Overwrite**
  
  True in order to overwrite the existing levels, False by default.

  **Type**    bool

**Example**
The following example shows how to overwrite the manufacturable values of an input parameter and how to define an additional value later.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
InputParameter = model.GetParameter(Name="P1")
InputParameter.UseManufacturableValues = True
InputParameter.AddLevels(Levels=["0.3 [mm]", "0.5 [mm]", "1e-3 [m]"], Overwrite=True)
InputParameter.AddLevels(Levels="7 [mm]")
```

CreateOptimizationCriterion

Creates an OptimizationCriterion entity associated to a parameter.

**Return**

The DataReference of the OptimizationCriterion.

**Type**    DataReference
Required Arguments

**Parameter**  The parameter on which the criterion is created.

**Type**  DataReference

**Example**
The following example shows how to create an OptimizationCriterion entity.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
parameter1 = model.GetParameter(Name="P1")
optimizationCriterion = parameter1.CreateOptimizationCriterion()
```

*DeleteDiscreteLevels*

Deletes a list of levels from a discrete input parameter.

**Required Arguments**

**DiscreteLevels**  List of the DiscreteLevel entities to delete.

**Type**  List<DataReference>

**Example**
The following example shows how to add and then delete one or more levels from a discrete input parameter.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DiscreteInputParameter = model.GetParameter(Name="P1")
level1 = DiscreteInputParameter.AddDiscreteLevel(Value=3000, DisplayText="Three thousand turns")
level2 = DiscreteInputParameter.AddDiscreteLevel(Value=2000, DisplayText="Two thousand turns")
level3 = DiscreteInputParameter.AddDiscreteLevel(Value=5000, Index="1")
DiscreteInputParameter.DeleteDiscreteLevels(DiscreteLevels=[level1, level2])
```

*DeleteLevels*

Deletes a list of levels from a continuous input parameter.

**Required Arguments**

**Indices**  Indices of the items to remove from the levels list

**Type**  List<int>

**Example**
The following example shows how to add and then delete one or more levels from a continuous input parameter for which the UseManufacturableValues property is set to True.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
InputParameter = model.GetParameter(Name="P1")
InputParameter.UseManufacturableValues = True
InputParameter.AddLevels(Levels=["0.3 [mmm]", "0.5 [mm]", "1e-3 [m]"], Overwrite=True)
InputParameter.DeleteLevels(Indices=[0, 1])
```
ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName

The exported file name.

Type string

Optional Arguments

AppendMode

True to append to an existing csv file, False to overwrite it.

Type bool

Default Value False

Example

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

GetDiscreteLevel

Get a discrete level by name from an input parameter. The parameter's full and ordered list of discrete levels is available as its "DiscreteLevels" property.

Return

The DataReference of the discrete level entity.

Type DataReference

Required Arguments

Name

The name of the discrete level.

Type string

Example

The following example shows how the user can retrieve a discrete level of a discrete input parameter by its name.

```
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
DiscreteInputParameter1 = dOEModel1.GetParameter(Name="P1")
level = DiscreteInputParameter1.GetDiscreteLevel(Name="Level 1")
```
GetOptimizationCriterion

Get the DataReference of the OptimizationCriterion associated with a Parameter. An exception is thrown if the entity is not found.

Return

The DataReference of the OptimizationCriterion.

Type

DataReference

Required Arguments

Parameter

Parent Parameter.

Type

DataReference

Example

The following example shows how the user can get a OptimizationCriterion to change one of its properties.

```python
system1 = GetSystem(Name="RSO")
optimization1 = system1.GetContainer(ComponentName="Optimization")
optimizationModel1 = optimization1.GetModel()
parameter3 = optimizationModel1.GetParameter(Name="P3")
optimizationCriterion = parameter3.GetOptimizationCriterion()
optimizationCriterion.ObjectiveType = "eGT_MinimumPossible"
```

GetParameterStatistics

Get the DataReference of the ParameterStatistics entity associated with a Parameter.

Return

The DataReference of the ParameterStatistics entity.

Type

DataReference

Example

The following example shows how the user can get a ParameterStatistics entity and examine its Mean property.

```python
system1 = GetSystem(Name="SSA")
sixSigmaAnalysis1 = system1.GetContainer(ComponentName="Six Sigma Analysis")
sixSigmaModel1 = sixSigmaAnalysis1.GetModel()
parameter4 = sixSigmaModel1.GetParameter(Name="P4")
parameterStatistics1 = parameter4.GetParameterStatistics()
mean = parameterStatistics1.Mean
```

MisqpOptimization

Entity which performs and manages the DesignXplorer Optimization Method Component

Properties

Converged

Convergence state

Type

bool
Read Only  Yes

**DerivativeApproximationType**

DerivativeApproximationType

**Type**  DerivativeApproximationType

Read Only  No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  string

Read Only  No

**MaxConvPercentage**

Allowable Convergence Percentage

**Type**  double

Read Only  No

**MaximumNumberOfPermutations**

Maximum number of permutations for discrete / manufacturable values.

**Type**  int

Read Only  Yes

**MaxNumCandidates**

Maximum Number of Candidates

**Type**  int

Read Only  No

**MaxNumIterations**

Maximum Number of Iterations

**Type**  int

Read Only  No

**NumberOfEvaluations**

Number of Evaluations

**Type**  int
**Read Only**  Yes

**NumberOfFailures**
Number of Failures
Type  int
Read Only  Yes

**NumCandidates**
Number of Candidates
Type  int
Read Only  Yes

**NumIterations**
Current Iteration
Type  int
Read Only  Yes

**SampleSetSize**
Size of Generated Sample Set
Type  int
Read Only  Yes

**MOGAOptimization**
Entity which performs and manages the DesignXplorer Optimization Method Component

**Properties**

**Converged**
Convergence state
Type  bool
Read Only  Yes

**CrossoverProbability**
Maximum Allowable Pareto Percentage
Type  double
Read Only  No
**CurrentParetoPercentage**

Pareto Percentage

- **Type**: double
- **Read Only**: Yes

**CurrentStabilityPercentage**

Stability Percentage

- **Type**: double
- **Read Only**: Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**MaximumNumberOfPermutations**

Maximum number of permutations for discrete / manufacturable values.

- **Type**: int
- **Read Only**: Yes

**MaxNumCandidates**

Maximum Number of Candidates

- **Type**: int
- **Read Only**: No

**MaxNumCycles**

Maximum Number Of Cycles for OSFD algorithm.

- **Type**: int
- **Read Only**: No

**MaxNumIterations**

Maximum Number of Iterations

- **Type**: int
- **Read Only**: No
**MutationProbability**

Maximum Allowable Pareto Percentage

*Type:* double

*Read Only:* No

**NumberOfEvaluations**

Number of Evaluations

*Type:* int

*Read Only:* Yes

**NumberOfFailures**

Number of Failures

*Type:* int

*Read Only:* Yes

**NumberOfInitialSamples**

Number of Initial Samples.

*Type:* int

*Read Only:* No

**NumCandidates**

Number of Candidates

*Type:* int

*Read Only:* Yes

**NumIterations**

Current Iteration

*Type:* int

*Read Only:* Yes

**NumSamplesPerIter**

Number of Samples Per Iteration

*Type:* int

*Read Only:* No
**ParetoPercentage**

Maximum Allowable Pareto Percentage

**Type**       double
**Read Only**  No

**RandomGeneratorSeed**

LHS Seed

**Type**       int
**Read Only**  No

**SampleSetSize**

Size of Generated Sample Set

**Type**       int
**Read Only**  Yes

**StabilityCriterion**

Stability Criterion

**Type**       double
**Read Only**  No

**TypeOfInitialSampling**

Type Of Initial Sampling

**Type**       TypeOfInitialSampling
**Read Only**  No

**NlpqlOptimization**

Entity which performs and manages the DesignXplorer Optimization Method Component

**Properties**

**Converged**

Convergence state

**Type**      bool
**Read Only** Yes
**DerivativeApproximationType**

DerivativeApproximationType

**Type** DerivativeApproximationType

**Read Only** No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

**Read Only** No

**MaxConvPercentage**

Allowable Convergence Percentage

**Type** double

**Read Only** No

**MaximumNumberOfPermutations**

Maximum number of permutations for discrete / manufacturable values.

**Type** int

**Read Only** Yes

**MaxNumCandidates**

Maximum Number of Candidates

**Type** int

**Read Only** No

**MaxNumIterations**

Maximum Number of Iterations

**Type** int

**Read Only** No

**NumberOfEvaluations**

Number of Evaluations

**Type** int

**Read Only** Yes
**NumberOfFailures**

Number of Failures

**Type** int

**Read Only** Yes

**NumCandidates**

Number of Candidates

**Type** int

**Read Only** Yes

**NumIterations**

Current Iteration

**Type** int

**Read Only** Yes

**SampleSetSize**

Size of Generated Sample Set

**Type** int

**Read Only** Yes

**OptimizationCriterion**

The data entity which describes the objective and constraint associated with a Parameter for an Optimization Study.

**Properties**

**ConstraintFirstValue**

Constraint First Value used when ConstraintType is eGI_LessThanTarget, eGI_GreaterThanTarget, eGI_NearTarget or eGI_InsideBounds.

**Type** double

**Read Only** No

**ConstraintHandling**

Constraint Handling

**Type** ConstraintHandlingType

**Read Only** No
**ConstraintImportance**

Importance of the constraint when multiple objectives are defined.

<table>
<thead>
<tr>
<th>Type</th>
<th>ImportanceLevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**ConstraintSecondValue**

Constraint Second Value used when ConstraintType is eGl_InsideBounds.

<table>
<thead>
<tr>
<th>Type</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**ConstraintType**

Constraint type for the Parameter to be optimized.

<table>
<thead>
<tr>
<th>Type</th>
<th>ConstraintType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**HistoryChart**

DataReference of the associated history chart entity.

<table>
<thead>
<tr>
<th>Type</th>
<th>DataReference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**LowerBound**

Lower Bound of the variation range if Parameter is an input parameter.

<table>
<thead>
<tr>
<th>Type</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**ObjectiveImportance**

Importance of the objective when multiple objectives are defined.

<table>
<thead>
<tr>
<th>Type</th>
<th>ImportanceLevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**ObjectiveTargetValue**

Objective Target Value if ObjectiveType is SeekTarget.

<table>
<thead>
<tr>
<th>Type</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>
**Objective Type**

Objective type for the Parameter to be optimized.

**Type**  
**GoalType**

**Read Only**  
**No**

**Starting Value**

The value of the parameter in the starting point if Parameter is an input parameter and the optimization method uses a starting point.

**Type**  
**double**

**Read Only**  
**No**

**Units**

Unit of the Parameter.

**Type**  
**string**

**Read Only**  
**Yes**

**Upper Bound**

Upper Bound of the variation range if Parameter is an input parameter.

**Type**  
**double**

**Read Only**  
**No**

**Methods**

**Get Chart**

Get the DataReference of the HistoryChart associated with an OptimizationCriterion. An exception is thrown if the entity is not found.

**Return**  
The DataReference of the OptimizationCriterion.

**Type**  
**DataReference**

**Example**

The following example shows how the user can get a HistoryChart to export its data as .csv file.

```plaintext
system = GetSystem(Name="RSO")
optimization = system.GetContainer(ComponentName="Optimization")
optimizationModel = optimization.GetModel()
parameter = optimizationModel.GetParameter(Name="P3")
criterion = parameter.GetOptimizationCriterion()
historyChart = criterion.GetChart()
historyChart.ExportData(FileName="D:/Temp/HistoryChart.csv")
```
**OptimizationMethod**

Entity which wraps and manages the external Optimization Method for the Optimization component

**Properties**

No Properties.

**OptimizationModel**

Entity which performs and manages the DesignXplorer Optimization Component

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type*  
string

*Read Only*  
No

**ExportDesignPoints**

If True and PreserveDesignPoints is True as well, export project for each preserved Design Points.

*Type*  
bool

*Read Only*  
No

**Method**

Optimization Method

*Type*  
DataReference

*Read Only*  
No

**MethodName**

Type of the optimization

*Type*  
string

*Read Only*  
No

**NumberOfRetries**

Indicates the number of times DX will try to update the failed design points.

*Type*  
int

*Read Only*  
No
**PreserveDesignPoints**

If True, preserve the Design Points at the project level after the component Update.

**Type**  
bool

**Read Only**  
No

**RetainDesignPoints**

If True and PreserveDesignPoints is True as well, retain data for each preserved Design Points.

**Type**  
bool

**Read Only**  
No

**RetryDelay**

Indicates how much time will elapse between tries. This option is only applicable when NumberOfRetries is greater than 0, otherwise it has no effect.

**Type**  
Quantity

**Read Only**  
No

**VerifyCandidatePoints**

If True, verifies the candidates by a design points update.

**Type**  
bool

**Read Only**  
No

**Methods**

**CreateChart**

Creates a Chart entity. The chart must be updated once after its creation by invoking its Update method. Then changes to its properties will update the chart automatically.

**Return**  
The DataReference of the Chart.

**Type**  
DataReference

**Required Arguments**

**ChartType**  
Type of chart to be created. The possible values depend on the type of Model. For instance, a ResponseSurface model accepts Spider, LocalSensitivity and Response chart while a CorrelationModel accepts CorrelationMatrix, DeterminationMatrix and CorrelationScatter charts.

**Type**  
ChartType

**Optional Arguments**

**DisplayText**  
Displayed name of the chart. If not specified, a default name is applied, depending on the value of ChartType.
**Type**  
`string`

**Example**  
The following example shows how to create a chart.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
TradeoffChart = model.CreateChart(ChartType="eChartTradeoff")
TradeoffChart.Update()
```

**CreateCustomCandidatePoint**

Creates a custom candidate point. The Expressions dictionary can be used to specify a value or a quantity for some or all of the input parameters. In the context of a response surface based optimization, the output values are evaluated from the response surface.

**Return**  
The created entity.

**Type**  
`DataReference`

**Optional Arguments**

**DisplayText**  
DisplayText of the created entity. If not specified, a default name of the form "Custom Candidate Point [#]" is used.

**Type**  
`string`

**Expressions**  
The values for each input parameter. If not specified, each parameter is initialized to its current value.

**Type**  
`IDictionary<DataReference, Object>`

**Example**  
The following example shows how to create a CustomCandidatePoint entity on an Optimization model based on a ResponseSurface.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
inputParameter1 = model.GetParameter(Name="P1")
inputParameter2 = model.GetParameter(Name="P2")
customCandidatePoint = model.CreateCustomCandidatePoint( DisplayText="Existing Design",
    Expressions={inputParameter1: "2.01", inputParameter2: "15.5"})
```

**CreateParameterRelationship**

Creates a Parameter Relationship

**Return**  
The DataReference of the ParameterRelationship.

**Type**  
`DataReference`

**Optional Arguments**

**LeftExpression**  
Left Expression.
**Design Exploration**

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RightExpression</strong></td>
<td>Right Expression.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>string</td>
</tr>
<tr>
<td><strong>Parameter Relationship Type</strong></td>
<td>Parameter Relationship Type.</td>
</tr>
<tr>
<td><strong>Default Value</strong></td>
<td>ePRT_LessThanOrEqualTo</td>
</tr>
</tbody>
</table>

**Example**
The following example shows how to create an ParameterRelationship entity.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
parameterRelationship = model.CreateParameterRelationship(LeftExpression="P1+P2", RightExpression="P3", Type="PRT_LessThanOrEqualTo")
parameterRelationship2 = model.CreateParameterRelationship(LeftExpression="P4+P5", RightExpression="10[inch]", Type="PRT_GreaterThanOrEqualTo")
```

**DeleteCharts**
Deletes a list of Chart entities.

**Required Arguments**

<table>
<thead>
<tr>
<th>Charts</th>
<th>List of Chart entities to delete.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>List&lt;DataReference&gt;</td>
</tr>
</tbody>
</table>

**Example**
The following example shows how to delete existing charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart1 = model.GetChart(Name="TradeoffChart 1")
chart2 = model.GetChart(Name="SamplesChart 1")
model.DeleteCharts(Charts=[chart1, chart2])
```

**DeleteCustomCandidatePoints**
Deletes a list of CustomCandidatePoint entities.

**Required Arguments**

<table>
<thead>
<tr>
<th>CustomCandidatePoints</th>
<th>CustomCandidatePoint entities to delete.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>List&lt;DataReference&gt;</td>
</tr>
</tbody>
</table>

**Example**
The following example shows how to delete existing CustomCandidatePoint entities from an optimization model.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
```
customCandidatePoint1 = model.GetCustomCandidatePoint(Name="CustomCandidatePoint 1")
customCandidatePoint2 = model.GetCustomCandidatePoint(Name="CustomCandidatePoint 2")
model.DeleteCustomCandidatePoints(CustomCandidatePoints=[customCandidatePoint1, customCandidatePoint2])

**DeleteOptimizationCriteria**

Deletes a list of OptimizationCriterion entities.

**Required Arguments**

**OptimizationCriteria** list of Objective entities to delete.

**Type** List<DataReference>

**Example**
The following example shows how to delete existing OptimizationCriterion entities.

container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
parameter1 = model.GetParameter(Name="P1")
optimizationCriterion1 = parameter1.GetOptimizationCriterion()
parameter3 = model.GetParameter(Name="P3")
optimizationCriterion3 = parameter3.GetOptimizationCriterion()
model.DeleteOptimizationCriteria(OptimizationCriteria=[optimizationCriterion1, optimizationCriterion3])

**DeleteParameterRelationships**

Deletes a list of ParameterRelationship entities.

**Required Arguments**

**ParameterRelationships** DataReferences of the entities to delete

**Type** List<DataReference>

**Example**
The following example shows how to delete existing ParameterRelationships.

container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
pr1 = model.GetParameterRelationship(Name="ParameterRelationShip 1")
pr2 = model.GetParameterRelationship(Name="ParameterRelationShip 2")
pr3 = model.GetParameterRelationship(Name="ParameterRelationShip 3")
model.DeleteParameterRelationships(ParameterRelationships=[pr1, pr2, pr3])

**DuplicateChart**

Duplicates a Chart entity. The chart must be updated once after its creation by invoking its Update method. Then changes to its properties will update the chart automatically.

**Return** The DataReference of the Chart.

**Type** DataReference
**Required Arguments**

**Chart**  
The source chart to duplicate.

  **Type**  *DataReference*

**Optional Arguments**

**DisplayText**  
Displayed name of the chart. If not specified, a default name is applied, depending on the value of ChartType.

  **Type**  *string*

**TargetModel**  
The model on which the duplicated chart is created. If this parameter is not set, the model of the source chart is used.

  **Type**  *DataReference*

**TargetResponsePoint**  
Parent TargetResponsePoint of the chart. If this parameter is not set, the response point of the source chart is used if applicable.

  **Type**  *DataReference*

**Example**  
The following example shows how to duplicate a chart.

```csharp
  designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
  dOEModel1 = designofExperiment1.GetModel()
  designPointsCurvesChart1 = dOEModel1.GetChart(Name="DesignPointsCurves")
  chart1 = dOEModel1.DuplicateChart(Chart=designPointsCurvesChart1)
  chart1.Update()
```

**ExportData**

Export the data of model's entities to a csv file. These entities can be candidate points, custom candidate points, charts, parametric tables or input parameters.

**Required Arguments**

**Entities**  
An optional list of entities containing data to be exported in the same file.

  **Type**  *List<DataReference>*

**FileName**  
The exported file name.

  **Type**  *string*

**Optional Arguments**

**AppendMode**  
True to append to an existing csv file, False to overwrite it.

  **Type**  *bool*

  **Default Value**  *False*
**Example**
The following example shows how the user can export the candidate points of an optimization component.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
candidates = model.GetCandidatePoints()
model.ExportData(FileName="doe.csv", Entities=candidates)
```

**GetCandidatePoint**
Get the DataReference of a CandidatePoint entity. An exception is thrown if the entity is not found.

**Return**
The DataReference of the CandidatePoint entity.

**Type**
DataReference

**Required Arguments**

**Name**
Name of the CandidatePoint to retrieve.

**Type**
string

**Example**
The following example shows how to retrieve an existing CandidatePoint entity.

```python
system1 = GetSystem(Name="DOP")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
candidatePoint = model.GetCandidatePoint(Name="CandidatePoint 1")
```

**GetCandidatePoints**
Get the DataReferenceSet of all of the existing CandidatePoint entities on the model.

**Return**
The DataReferenceSet of the CandidatePoint entities.

**Type**
DataReferenceSet

**Example**
The following example shows how to retrieve the candidate points of an optimization model.

```python
system1 = GetSystem(Name="DOP")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
candidatePoints = model.GetCandidatePoints()
```

**GetChart**
Query to return the chart reference for a given model and chart name.

**Return**
The DataReference of the chart.

**Type**
DataReference
Required Arguments

Name  Name of the chart.
   Type  string

GetConvergenceChart

Query to return the chart reference for a given model and chart name.

Return  The DataReference of the chart.
   Type  DataReference

GetCustomCandidatePoint

Get the DataReference of a CustomCandidatePoint entity. An exception is thrown if the entity is not found.

Return  The DataReference of the CustomCandidatePoint entity.
   Type  DataReference

Required Arguments

Name  Name of the CustomCandidatePoint to retrieve.
   Type  string

Example

The following example shows how to retrieve an existing CustomCandidatePoint entity.

```python
system1 = GetSystem(Name="DOP")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
customCandidatePoint = model.GetCustomCandidatePoint(Name="CustomCandidatePoint 1")
```

GetCustomCandidatePoints

Get the DataReferenceSet of all of the existing CustomCandidatePoint entities of the model.

Return  The DataReferenceSet of the CustomCandidatePoint entities.
   Type  DataReferenceSet

Example

The following example shows how to retrieve the custom candidate points of an optimization model.

```python
system1 = GetSystem(Name="DOP")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
customCandidatePoints = model.GetCustomCandidatePoints()
```
**GetOptimizationCriteria**

Get the DataReferenceSet of all of the existing OptimizationCriterion entities of the model.

**Return**  
The DataReference of the OptimizationCriterion entities.

**Type**  
DataReferenceSet

**Required Arguments**

**Model**  
Parent Optimization model

**Type**  
DataReference

**Example**

The following example shows how to retrieve the objectives of an optimization model, as a set of OptimizationCriterion entities.

```python
system1 = GetSystem(Name="DOP")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
criteria = model.GetOptimizationCriteria()
```

**GetParameter**

Get the DataReference of a Parameter. An exception is thrown if the entity is not found.

**Return**  
The DataReference of the Parameter.

**Type**  
DataReference

**Required Arguments**

**Name**  
Name of the Parameter.

**Type**  
string

**Example**

The following example shows how the user can get a parameter of a model to change one of its properties.

```python
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
inputParameter1 = dOEModel1.GetParameter(Name="P1")
inputParameter1.LowerBound = 1
```

**GetParameterRelationship**

Get the DataReference of a ParameterRelationship entity. An exception is thrown if the entity is not found.

**Return**  
The DataReference of the ParameterRelationship entity.

**Type**  
DataReference
Required Arguments

**Name**  Name of the ParameterRelationship to retrieve.

**Type**  string

**Example**  
The following example shows how the user can retrieve an existing ParameterRelationship entity.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
parameterRelationship = model.GetParameterRelationship(Name="ParameterRelationShip 1")
```

**GetParameterRelationships**  
Get the Parameter Relationships associated to a Model. If the optional argument Enabled is not specified, the query returns all Parameter Relationships. If it is specified and True, the query returns only enabled Parameter Relationships. If it is False, the query returns only disabled Parameter Relationships.

**Return**  
The DataReferenceSet of the ParameterRelationship entities.

**Type**  DataReferenceSet

**Optional Arguments**

**Enabled**  
If True, the query returns only enabled Parameter Relationships. If False, the query returns only disabled Parameter Relationships. If the argument is omitted, the query returns all Parameter Relationships.

**Type**  bool

**Example**  
The following example shows how the user can retrieve all the Parameter Relationships defined on a model.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
allParameterRelationships = model.GetParameterRelationships()
enabledParameterRelationships = model.GetParameterRelationships(Enabled="True")
disabledParameterRelationships = model.GetParameterRelationships(Enabled="False")
```

**GetParameters**  
Get the DataReferences of the InputParameter and OutputParameter of the model. If the optional argument InputParameters is not specified, the query returns all parameters. If it is specified and True, the query returns only input parameters. If it is False, the query returns only output parameters.

**Return**  
The DataReferences of the Parameters

**Type**  DataReferenceSet

**Optional Arguments**

**InputParameters**  
If True, the query returns only input parameters. If False, the query returns only output parameters. If the argument is omitted, the query returns all parameters.
Type  bool

Example

The following example shows how the user can get the parameters of a model.

```python
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
parameters = dOEModel1.GetParameters()
```

GetParametricTable

Get the DataReference of ParametricTable. An exception is thrown if the entity is not found. Names of the tables generated internally are: "DesignPoints", "CorrelationMatrix", "CorrelationScatter", "MinDesignPoints", "MaxDesignPoints", "ResponsePoints", "DeterminationMatrix".

Return  The DataReference of the ParametricTable

Type  DataReference

Required Arguments

Name  Name of the ParametricTable

Type  string

Example

The following example shows how the user can get a ParametricTable to add a new row and set values.

```python
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
parametricTable = dOEModel1.GetParametricTable(Name="DesignPoints")
parametricTable.AddRow()
parametricTable.SetCellValue(RowIndex=9, ColumnIndex=0, Value="2.1")
```

SetVariationReferencePoint

Sets a point as the reference to calculate the variation of the parameters in each of the candidate and custom candidate points of the container. The variation is calculated by the command and can be retrieved for each parameter of each point.

Required Arguments

Point  Point entity to set as the reference point.

Type  DataReference

Optional Arguments

Source  The source of the output values to set as the reference.

Type  OutputSource

Default Value  Simulation
Example
The following example shows how to set a custom candidate point as the reference and how to retrieve the calculated variation on the parameters of a second candidate point.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
customCandidatePoint = model.GetCustomCandidatePoint(Name="CustomCandidatePoint")
model.SetVariationReferencePoint(Point=customCandidatePoint, Source="Simulation")
bestCandidate = model.GetCandidatePoint(Name="CandidatePoint")
variationOfOutputsAsDecimals = bestCandidate.GetOutputValues(Source="Simulation", ValueType="VariationToReference")
```

VerifyPoints
Verify the CandidatePoint entities by performing a design point update to generate Simulation output values, so that they can be compared to the ResponseSurface output values. This command has no effect if the optimization model is used as part of a Direct Optimization system.

Required Arguments
Points List of points to verify.

Type List<DataReference>

Optional Arguments
PullFromCacheOnly If True, the command attempts to pull output values from the design point cache but does not trigger any design point update.

Type bool

Default Value False

Example
The following example shows how the user can verify all the candidate points generated by the optimization model and extract the existing output parameter values from the first candidate point.

```python
system1 = GetSystem(Name="RSO")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
candidatePoints = model.GetCandidatePoints()
model.VerifyPoints(Points=candidatePoints)
responseSurfaceValues = candidatePoints[0].GetOutputValues(Source="ResponseSurface")
verifiedValues = candidatePoints[0].GetOutputValues(Source="Simulation")
```

OutputParameter
Output parameter entity for DesignXplorer.

Properties

DisplayText
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.
Type string
Read Only No

**FTestFiltering**

F-Test Filtering (beta)

Type bool
Read Only No

**InheritFromModelSettings**

Determines whether the Maximum Predicted Relative Error defined at the Model level is applicable to the output parameter.

Type bool
Read Only No

**LowerBound**

Minimum value extracted from existing design points and/or sample sets.

Type double
Read Only Yes

**MaxPredictedRelativeError**

Maximum Relative Error targeted for an output parameter when refining with the Kriging algorithm. This is the maximum predicted relative error that is acceptable for the selected output parameter.

Type double
Read Only No

**PredictedRelativeError**

Current value of the Predicted Relative Error when refining with the Kriging algorithm

Type double
Read Only Yes

**TransformationType**

Transformation Type

Type TransformationType
Read Only No
**Units**

Units

**Type**  
string

**Read Only**  
Yes

**UpperBound**

Maximum value extracted from existing design points and/or sample sets.

**Type**  
double

**Read Only**  
Yes

**Methods**

**CreateOptimizationCriterion**

Creates an OptimizationCriterion entity associated to a parameter.

**Return**  
The DataReference of the OptimizationCriterion.

**Type**  
DataReference

**Required Arguments**

**Parameter**  
The parameter on which the criterion is created.

**Type**  
DataReference

**Example**

The following example shows how to create an OptimizationCriterion entity.

```plaintext
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
parameter1 = model.GetParameter(Name="P1")
optimizationCriterion = parameter1.CreateOptimizationCriterion()
```

**GetOptimizationCriterion**

Get the DataReference of the OptimizationCriterion associated with a Parameter. An exception is thrown if the entity is not found.

**Return**  
The DataReference of the OptimizationCriterion.

**Type**  
DataReference

**Required Arguments**

**Parameter**  
Parent Parameter.

**Type**  
DataReference
Example
The following example shows how the user can get a OptimizationCriterion to change one of its properties.

```java
system1 = GetSystem(Name="RSO")
optimization1 = system1.GetContainer(ComponentName="Optimization")
optimizationModel1 = optimization1.GetModel()
parameter3 = optimizationModel1.GetParameter(Name="P3")
optimizationCriterion = parameter3.GetOptimizationCriterion()
optimizationCriterion.ObjectiveType = "eGT_MinimumPossible"
```

GetParameterStatistics
Get the DataReference of the ParameterStatistics entity associated with a Parameter.

Return The DataReference of the ParameterStatistics entity.

Type DataReference

Example
The following example shows how the user can get a ParameterStatistics entity and examine its Mean property.

```java
system1 = GetSystem(Name="SSA")
sixSigmaAnalysis1 = system1.GetContainer(ComponentName="Six Sigma Analysis")
sixSigmaModel1 = sixSigmaAnalysis1.GetModel()
parameter4 = sixSigmaModel1.GetParameter(Name="P4")
parameterStatistics1 = parameter4.GetParameterStatistics()
mean = parameterStatistics1.Mean
```

ParameterRelationship
Entity which manages the calculated Statistics of a parameter for a Six-Sigma Component

Properties

Enabled
True if the Parameter is enabled for the current study.

Type bool

ReadOnly No

ErrorMessage
Error Message associated to the Parameter Relationship.

Type string

ReadOnly Yes

LeftExpression
Left Expression of Parameter Relationship.
**Type**
string

**Read Only**
No

**LeftExpressionQuantityName**
Quantity Name of Left Expression of Parameter Relationship.

**Type**
string

**Read Only**
No

**LeftExpressionValue**
Value of the left expression (as a quantity string), or an error message.

**Type**
string

**Read Only**
Yes

**RelationshipType**
Type of Parameter Relationship.

**Type**
ParameterRelationshipType

**Read Only**
No

**RightExpression**
Right Expression of Parameter Relationship.

**Type**
string

**Read Only**
No

**RightExpressionQuantityName**
Quantity Name of Right Expression of Parameter Relationship.

**Type**
string

**Read Only**
No

**RightExpressionValue**
Value of the right expression (as a quantity string), or an error message.

**Type**
string

**Read Only**
Yes
**Methods**

**DuplicateParameterRelationship**

Duplicates a Parameter Relationship.

*Return*  
The created entity.

*Type*  
DataReference

**Optional Arguments**

**TargetModel**  
The model on which the Parameter Relationship is created. If this parameter is not set, the model of the source Parameter Relationship is used.

*Type*  
DataReference

**Example**

The following example shows how to duplicate a Parameter Relationship.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
pr1 = model.GetParameterRelationship(Name="ParameterRelationShip")
pr2 = pr1.DuplicateParameterRelationship()
container = system1.GetContainer(ComponentName="Optimization 1")
model2 = container.GetModel()
pr3 = pr1.DuplicateParameterRelationship(TargetModel = model2)
```

**GetChart**

Get the DataReference of the ParameterRelationshipChart associated with a ParameterRelationship. An exception is thrown if the entity is not found.

*Return*  
The DataReference of the ParameterRelationshipChart.

*Type*  
DataReference

**Example**

The following example shows how the user can get a ParameterRelationshipChart to export its data as .csv file.

```python
system1 = GetSystem(Name="RSO")
optimization1 = system1.GetContainer(ComponentName="Optimization")
optimizationModel1 = optimization1.GetModel()
parameterRelationship1 = optimizationModel1.GetParameterRelationship(Name="ParameterRelationship")
chart1 = parameterRelationship1.GetChart()
Parameters.ExportData(Data=chart1,FileName="E:/Temp/Relationship.csv")
```

**ParameterRelationshipChart**

The data entity which describes a ParameterRelationship chart. It allows you to visualize how parameter relationship evolves during the optimization process, depending on the optimizer.
Properties

DisplayParameterFullName

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

Type  bool
Read Only  No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type  string
Read Only  No

IsUpToDate

True if the entity is up-to-date.

Type  bool
Read Only  No

ScreeningOptimization

Entity which performs and manages the DesignXplorer Optimization Method Component

Properties

Converged

Convergence state

Type  bool
Read Only  Yes

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type  string
Read Only  No

MaximumNumberOfPermutations

Maximum number of permutations for discrete / manufacturable values.

Type  int
**Read Only** Yes

**MaxNumCandidates**
Maximum Number of Candidates

**Type** int
**Read Only** No

**NumberOfEvaluations**
Number of Evaluations

**Type** int
**Read Only** Yes

**NumberOfFailures**
Number of Failures

**Type** int
**Read Only** Yes

**NumberOfSamples**
Number of Samples

**Type** int
**Read Only** No

**NumCandidates**
Number of Candidates

**Type** int
**Read Only** Yes

**SampleSetSize**
Size of Generated Sample Set

**Type** int
**Read Only** Yes
**DX Evaluation Container**

This container holds data for an instance of the Evaluation.

**Data Entities**

**CandidatePoint**

The CandidatePoint data entity is a candidate point automatically generated by an optimization.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**Methods**

**CanUpdate**

Returns true if the entity can be updated to produce the specific source of output values. As an example, the query will return false for Source=ResponseSurface if the candidate point was generated by a Direct Optimization system.

- **Return**: Results of the query telling if the Point entity can be updated to produce the specified source of output values.

- **Type**: bool

**Optional Arguments**

**Source**

Source of the output values to query for.

- **Type**: OutputSource
- **Default Value**: Simulation

**Example**

The following example shows how to check if a CandidatePoint entity can be updated and how to update it to obtain the Simulation output values.

```python
system1 = GetSystem(Name="RSO")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
candidatePoint = model.GetCandidatePoint(Name="CandidatePoint 1")
if candidatePoint.CanUpdate(Source="Simulation"):
    candidatePoint.Update(Source="Simulation")
```
**GetInputValues**

Get the values of the input parameters. Depending on the parameter definition, each value can be a Quantity, a real, a string or an integer.

**Return**

The dictionary of the input parameter values.

**Type**

ReadOnlyDictionary<DataReference, Object>

**Optional Arguments**

**ValueType**

The type of parameter value to return.

**Type**

ParameterValueType

**Default Value**

ActualValue

**Example**

The following example shows how to retrieve a candidate point and then extract its input parameter values.

```python
system1 = GetSystem(Name="DOP")
candidatePoint = model.GetCandidatePoint(Name="CandidatePoint 1")
dictInputValues = candidatePoint.GetInputValues()
```

**GetOutputValues**

Get the values of the output parameters for the specified source of output values.

**Return**

The dictionary of the output parameter values.

**Type**

ReadOnlyDictionary<DataReference, Object>

**Optional Arguments**

**Source**

The source of the output values to return.

**Type**

OutputSource

**Default Value**

Simulation

**ValueType**

The type of parameter value to return.

**Type**

ParameterValueType

**Default Value**

ActualValue

**Example**

The following example shows how to retrieve a candidate point and then extract its output parameter values.

```python
system1 = GetSystem(Name="DOP")
candidatePoint = model.GetCandidatePoint(Name="CandidatePoint 1")
dictOutputValues = candidatePoint.GetOutputValues()
```
candidatePoint = model.GetCandidatePoint(Name="CandidatePoint 1")
responseSurfaceOutputValues = candidatePoint.GetOutputValues(Source="ResponseSurface")
simulationOutputValues = candidatePoint.GetOutputValues(Source="Simulation")

**GetState**

Returns the state of the entity for the specified source of output values. A different state is available for each source of output values.

*Return*  
State for the specified nature of output values.  
*Type*  
**UpdatableEntityState**

**Optional Arguments**

**Source**  
Source of output values to query for.  
*Type*  
**OutputSource**  
*Default Value*  
Simulation

**Example**

The following example shows how to check the state of the Simulation's source of the output values of a CandidatePoint entity.

```python
system1 = GetSystem(Name="RSO")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
candidatePoint = model.GetCandidatePoint(Name="CandidatePoint 1")
state = candidatePoint.GetState(Source="Simulation")
```

**GetValue**

Get the value for a given parameter and, if the parameter is an output parameter, a given source of output values. Depending on the parameter definition, each value can be a Quantity, a real, a string or an integer.

*Return*  
The retrieve parameter value.  
*Type*  
**Object**

**Required Arguments**

**Parameter**  
The Parameter for which the value is requested.  
*Type*  
**DataReference**

**Optional Arguments**

**Source**  
If Parameter is an output parameter, the source of the output value to return.  
*Type*  
**OutputSource**  
*Default Value*  
Simulation

**ValueType**  
The type of parameter value to return.
**Type** ParameterValueType

**Default Value** ActualValue

**Example**

The following example shows how to retrieve a candidate point generated by the optimization model, and then extract selected parameter values.

```python
system1 = GetSystem(Name="DOP")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
candidatePoint = model.GetCandidatePoint(Name="CandidatePoint 1")
inputParam1 = model.GetParameter(Name="P1")
parameterValue1 = candidatePoint.GetValue(Parameter=inputParam1)
outputParam4 = model.GetParameter(Name="P4")
parameterValue2 = candidatePoint.GetValue(Parameter=outputParam4, Source="Simulation")
print parameterValue2.Value
```

**GetValues**

Get the values of all input parameters and the values of all output parameters for the specified source of output values. Depending on the parameter definition, each value can be a Quantity, a real, a string or an integer.

**Return**

The dictionary of the parameter values.

**Type** ReadOnlyDictionary<DataReference, Object>

**Optional Arguments**

**Source** For output parameters, the source of the output values to return.

**Type** OutputSource

**Default Value** Simulation

**Value Type** The type of parameter value to return.

**Type** ParameterValueType

**Default Value** ActualValue

**Example**

The following example shows how to retrieve a candidate point and then extract all of its parameter values.

```python
system1 = GetSystem(Name="DOP")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
candidatePoint = model.GetCandidatePoint(Name="CandidatePoint 1")
parameterValues = candidatePoint.GetValues(Source="Simulation")
```

**Update**

Updates a point entity to provide the output parameter values of the requested source. If the requested source is ResponseSurface, the output values are generated by evaluating the response surface model,
if available. If the requested source is Simulation, the output values are generated by triggering a DesignPoint update. This command applies to CandidatePoint and CustomCandidatePoint entities.

**Return**

The dictionary of parameters with their values.

**Type**  
ReadOnlyDictionary<TDataReference, TObject>

**Optional Arguments**

**Source**  
Source of output values to produce. The update method depends on this parameter.

**Type**  
OutputSource

**Default Value**  
Simulation

**Example**

The following example shows how to update an existing candidate point to obtain Simulation output values.

```python
system1 = GetSystem(Name="RSO")
candidate = system1.GetContainer(ComponentName="Optimization")
candidate1 = container.GetCandidatePoint(Name="CandidatePoint 1")
outputValues = candidate1.Update(Source="Simulation")
```

**CandidatesChart**

The data entity which describes the Candidate Points chart. The Candidate Points chart allows you to view different kinds of information about candidate points. It allows you specify one or more parameters for which you want to display candidates data. Color-coding and a legend make it easy to view and interpret samples, candidate points identified by the optimization, candidates inserted manually, and candidates for which output values have been verified by a design point update. You can specify the chart's properties to control the visibility of each axis, feasible samples, candidates you've inserted manually, and candidates with verified output values.

**Properties**

**CandidatesColoringMethod**

Coloring method used to draw the chart.

**Type**  
CandidatesColoringMethods

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

**Type**  
bool

**Read Only**  
No
**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**IsUpToDate**

True if the entity is up-to-date.

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**ShowCandidates**

If True, the candidates are displayed on the chart.

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**ShowCustomCandidates**

If True, any custom candidates are displayed on the chart.

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**ShowSamples**

If True, the samples are displayed on the chart.

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**ShowStartingPoint**

If True, the starting point is displayed on the chart.

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**ShowVerifiedCandidates**

If True, any verified candidates are displayed on the chart.

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>
Methods

EnableParameters

Enable or disable a list of parameters in a chart.

Required Arguments

isEnabled     False to disable the parameters, or True (default) to enable them.

    Type   bool

Optional Arguments

Parameters    Parameters to enable or disable, or all parameters if not specified.

    Type   List<DataReference>

Example

The following example shows how to disable two parameters, and then how to enable all parameters by omitting the Parameters optional parameter. This example uses a Correlation Matrix chart. The method also applies for other types of charts like LocalSensitivity, SpiderChart, etc.

```csharp
container = system1.GetContainer(ComponentName="Correlation")
model = container.GetModel()
param1 = model.GetParam(Name="P1")
param4 = model.GetParam(Name="P4")
chart = model.GetChart(Name="Correlation Matrix 1")
chart.EnableParameters(Parameters=[param1;param4], IsEnabled=false )
chart.EnableParameters()
```

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName     The exported file name.

    Type   string

Optional Arguments

AppendMode    True to append to an existing csv file, False to overwrite it.

    Type   bool

    Default Value   False

Example

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```csharp
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)

Update
Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

Example
The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
Methods

EnableVariable

Enable or disable a variable in a chart. This command is currently limited to the CorrelationScatter chart, where LinearTrendLine_Variable and QuadraticTrendLine_Variable are the two eligible variables, and the ConvergenceCriteria chart.

Required Arguments

isEnabled  
False to disable the variable, or True (default) to enable it.  

Type  bool

Optional Arguments

Variable  
DataReference of the variable to enable or disable.  

Type  DataReference

Example

The following example shows how to disable and enable a variable in a CorrelationScatter or ConvergenceCriteria chart.

```c
container = system1.GetContainer(ComponentName="Correlation")
model = container.GetModel()
chart = model.GetChart(Name="CorrelationScatter 1")
chart.EnableVariable( Variable=chart.LinearTrendLine_Variable, IsEnabled=false)
chart.EnableParameter( Variable=chart.QuadraticTrendLine_Variable )
```

EnableVariables

Enable or disable a list of variables in a chart.

Required Arguments

isEnabled  
False to disable the variable, or True (default) to enable it.  

Type  bool

Optional Arguments

Variables  
DataReference of the variable to enable or disable.  

Type  List<DataReference>

Example

The following example shows how to disable two variables, and then how to enable all variables by omitting the Parameters optional variable. This example uses a CorrelationScatter chart. The method also applies for ConvergenceCriteria Chart.

```c
container = system1.GetContainer(ComponentName="Correlation")
model = container.GetModel()
var1 = Graphics.GetVariableXY(Name="MyName")
chart = model.GetChart(Name="CorrelationScatter 1")
chart.EnableVariables( Variables = [var1;var2], IsEnabled=false)
```
ConvergenceCurvesChart

The data entity which describes a Convergence Curves chart. It allows you to visualize the convergence of an algorithm: with the steps of the convergence as X-axis and the criteria of the convergence as Y-axis. This chart can display several curves based on the same steps of convergence.

Properties

DisplayParameterFullName

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

Type bool
Read Only No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type string
Read Only No

IsUpToDate

True if the entity is up-to-date.

Type bool
Read Only No

Methods

EnableParameters

Enable or disable a list of parameters in a chart.

Required Arguments

IsEnabled False to disable the parameters, or True (default) to enable them.

Type bool

Optional Arguments

Parameters Parameters to enable or disable, or all parameters if not specified.

Type List<DataReference>
Example
The following example shows how to disable two parameters, and then how to enable all parameters by omitting the Parameters optional parameter. This example uses a Correlation Matrix chart. The method also applies for other types of charts like LocalSensitivity, SpiderChart, etc.

```python
container = system1.GetContainer(ComponentName="Correlation")
model = container.GetModel()
param1 = model.GetParam(Name="P1")
param4 = model.GetParam(Name="P4")
chart = model.GetChart(Name="Correlation Matrix 1")
chart.EnableParameters(Parameters=[param1,param4], IsEnabled=False)
chart.EnableParameters()
```

ExportData
Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

**FileName**  
The exported file name.

**Type**  
string

Optional Arguments

**AppendMode**  
True to append to an existing csv file, False to overwrite it.

**Type**  
bool

**Default Value**  
False

Example
The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

Update
Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

Example
The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
```
**CorrelationMatrixChart**

The data entity which describes a Correlation Matrix chart. This matrix allows the user to visualize how the input and output parameters are coupled.

**Properties**

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

*Type* `bool`

*Read Only* No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type* `string`

*Read Only* No

**IsUpToDate**

True if the entity is up-to-date.

*Type* `bool`

*Read Only* No

**Methods**

**EnableParameters**

Enable or disable a list of parameters in a chart.

**Required Arguments**

**IsEnabled**  False to disable the parameters, or True (default) to enable them.

*Type* `bool`

**Optional Arguments**

**Parameters**  Parameters to enable or disable, or all parameters if not specified.

*Type* `List<DataReference>`
**Example**

The following example shows how to disable two parameters, and then how to enable all parameters by omitting the Parameters optional parameter. This example uses a Correlation Matrix chart. The method also applies for other types of charts like LocalSensitivity, SpiderChart, etc.

```python
container = system1.GetContainer(ComponentName="Correlation")
model = container.GetModel()
param1 = model.GetParam(Name="P1")
param4 = model.GetParam(Name="P4")
chart = model.GetChart(Name="Correlation Matrix 1")
chart.EnableParameters(Parameters=[param1,param4], isEnabled=False)
chart.EnableParameters()
```

**ExportData**

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

**Required Arguments**

- **FileName**  
  The exported file name.

  - **Type**  
    string

**Optional Arguments**

- **AppendMode**  
  True to append to an existing csv file, False to overwrite it.

  - **Type**  
    bool
  - **Default Value**  
    False

**Example**

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

**Update**

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

**Example**

The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
```
**CorrelationScatterChart**

The data entity which describes a Correlation Scatter chart. This scatter chart allows the user to visualize the samples used to compute the correlation for the selected parameter pair, as well as the linear and the quadratic trend lines.

**Properties**

**Axes**

Dictionary of the parameters associated to axes.

*Type*  
Dictionary<ChartAxes, DataReference>

*Read Only*  
Yes

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

*Type*  
bool

*Read Only*  
No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type*  
string

*Read Only*  
No

**IsUpToDate**

True if the entity is up-to-date.

*Type*  
bool

*Read Only*  
No

**LinearTrendLine_Variable**

The DataReference of the LinearTrendLine variable.

*Type*  
DataReference

*Read Only*  
No

**QuadraticTrendLine_Variable**

The DataReference of the QuadraticTrendLine variable.
**Type**  
DataReference

**Read Only**  
No

**Methods**

**AssociateParameterToAxis**

Associates a Parameter to the specified axis of the chart. The Parameter argument can be omitted which means that the axis is not set.

**Required Arguments**

**Axis**  
Axis to modify.

  - **Type**  
  ChartAxes

**Optional Arguments**

**Parameter**  
Parameter entity to be assigned to the specified axis.

  - **Type**  
  DataReference

**Example**
The following example shows how to assign parameters to the axes of a DesignPointsCurves chart.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
inputParam1 = model.GetParam(Name="P1")
outputParam1 = model.GetParam(Name="P5")
outputParam2 = model.GetParam(Name="P6")
chart = model.GetChart(Name="Design Point vs Parameter 1")
chart.AssociateParameterToAxis(Parameter=inputParam1, Axis="XAxis")
chart.AssociateParameterToAxis(Parameter=outputParam1, Axis="YAxis")
chart.AssociateParameterToAxis(Parameter=outputParam2, Axis="YRightAxis")
```

---

**EnableVariable**

Enable or disable a variable in a chart. This command is currently limited to the CorrelationScatter chart, where LinearTrendLine_Variable and QuadraticTrendLine_Variable are the two eligible variables, and the ConvergenceCriteria chart.

**Required Arguments**

**IsEnabled**  
False to disable the variable, or True (default) to enable it.

  - **Type**  
  bool

**Optional Arguments**

**Variable**  
DataReference of the variable to enable or disable.

  - **Type**  
  DataReference

**Example**
The following example shows how to disable and enable a variable in a CorrelationScatter or ConvergenceCriteria chart.
container = system1.GetContainer(ComponentName="Correlation")
model = container.GetModel()
chart = model.GetChart(Name="CorrelationScatter 1")
chart.EnableVariable( Variable=chart.LinearTrendLine_Variable, IsEnabled=false)
chart.EnableParameter( Variable=chart.QuadraticTrendLine_Variable )

**ExportData**

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

**Required Arguments**

**FileName**  
The exported file name.  

**Type**  
string

**Optional Arguments**

**AppendMode**  
True to append to an existing csv file, False to overwrite it.  

**Type**  
bool  

**Default Value**  
False

**Example**

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```plaintext
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

**Update**

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

**Example**

The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```plaintext
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

**CustomCandidatePoint**

The CustomCandidatePoint data entity is a candidate point created and edited by the user for comparison with the results of an optimization.
Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type            string
Read Only       No

Methods

CanUpdate

Returns true if the entity can be updated to produce the specific source of output values. As an example, the query will return false for Source=ResponseSurface if the candidate point was generated by a Direct Optimization system.

Return

Results of the query telling if the Point entity can be updated to produce the specified source of output values.

Type            bool

Optional Arguments

Source

Source of the output values to query for.

Type            OutputSource
Default Value   Simulation

Example

The following example shows how to check if a CandidatePoint entity can be updated and how to update it to obtain the Simulation output values.

```python
system1 = GetSystem(Name="RSO")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
candidatePoint = model.GetCandidatePoint(Name="CandidatePoint 1")
if candidatePoint.CanUpdate(Source="Simulation"):
    candidatePoint.Update(Source="Simulation")
```

GetInputValues

Get the values of the input parameters. Depending on the parameter definition, each value can be a Quantity, a real, a string or an integer.

Return

The dictionary of the input parameter values.

Type            ReadOnlyDictionary<DataReference, Object>

Optional Arguments

ValueType

The type of parameter value to return.
<table>
<thead>
<tr>
<th><strong>Type</strong></th>
<th><strong>ParameterValue</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Value</strong></td>
<td><strong>ActualValue</strong></td>
</tr>
</tbody>
</table>

**Example**
The following example shows how to retrieve a candidate point and then extract its input parameter values.

```python
system1 = GetSystem(Name="DOP")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
candidatePoint = model.GetCandidatePoint(Name="CandidatePoint 1")
dictInputValues = candidatePoint.GetInputValues()
```

**GetOutputValues**
Get the values of the output parameters for the specified source of output values.

**Return**
The dictionary of the output parameter values.

**Type**
`ReadOnlyDictionary<DataReference, Object>`

**Optional Arguments**

**Source**
The source of the output values to return.

**Type**
`OutputSource`

**Default Value**
`Simulation`

**ValueType**
The type of parameter value to return.

**Type**
`ParameterValue`

**Default Value**
`ActualValue`

**Example**
The following example shows how to retrieve a candidate point and then extract its output parameter values.

```python
system1 = GetSystem(Name="DOP")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
candidatePoint = model.GetCandidatePoint(Name="CandidatePoint 1")
responseSurfaceOutputValues = candidatePoint.GetOutputValues(Source="ResponseSurface")
simulationOutputValues = candidatePoint.GetOutputValues(Source="Simulation")
```

**GetState**
Returns the state of the entity for the specified source of output values. A different state is available for each source of output values.

**Return**
State for the specified nature of output values.

**Type**
`UpdatableEntityState`
Optional Arguments

**Source**  
Source of output values to query for.

<table>
<thead>
<tr>
<th>Type</th>
<th>OutputSource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>Simulation</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how to check the state of the Simulation's source of the output values of a CandidatePoint entity.

```python
system1 = GetSystem(Name="RSO")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
candidatePoint = model.GetCandidatePoint(Name="CandidatePoint 1")
state = candidatePoint.GetState(Source="Simulation")
```

**GetValue**

Get the value for a given parameter and, if the parameter is an output parameter, a given source of output values. Depending on the parameter definition, each value can be a Quantity, a real, a string or an integer.

**Return**

The retrieve parameter value.

<table>
<thead>
<tr>
<th>Type</th>
<th>Object</th>
</tr>
</thead>
</table>

**Required Arguments**

**Parameter**

The Parameter for which the value is requested.

<table>
<thead>
<tr>
<th>Type</th>
<th>DataReference</th>
</tr>
</thead>
</table>

**Optional Arguments**

**Source**

If Parameter is an output parameter; the source of the output value to return.

<table>
<thead>
<tr>
<th>Type</th>
<th>OutputSource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>Simulation</td>
</tr>
</tbody>
</table>

**ValueType**

The type of parameter value to return.

<table>
<thead>
<tr>
<th>Type</th>
<th>ParameterValueType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>ActualValue</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how to retrieve a candidate point generated by the optimization model, and then extract selected parameter values.

```python
system1 = GetSystem(Name="DOP")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
candidatePoint = model.GetCandidatePoint(Name="CandidatePoint 1")
```
inputParam1 = model.GetParameter(Name="P1")
parameterValue1 = candidatePoint.GetValue(Parameter=inputParam1)
outputParam4 = model.GetParameter(Name="P4")
parameterValue2 = candidatePoint.GetValue(Parameter=outputParam4, Source="Simulation")
print parameterValue2.Value

GetValues

Get the values of all input parameters and the values of all output parameters for the specified source of output values. Depending on the parameter definition, each value can be a Quantity, a real, a string or an integer.

**Return**

The dictionary of the parameter values.

**Type**

ReadOnlyDictionary<DataReference, Object>

Optional Arguments

**Source**

For output parameters, the source of the output values to return.

**Type**

OutputSource

**Default Value**

Simulation

**ValueType**

The type of parameter value to return.

**Type**

ParameterValueType

**Default Value**

ActualValue

Example

The following example shows how to retrieve a candidate point and then extract all of its parameter values.

```python
system1 = GetSystem(Name="DOP")
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
candidatePoint = model.GetCandidatePoint(Name="CandidatePoint 1")
parameterValues = candidatePoint.GetValues(Source="Simulation")
```

SetParameter

Sets the expression of a parameter in a CustomCandidatePoint entity. In the context of a response surface based optimization, the output values are updated from the response surface.

Required Arguments

**Expression**

Assigned Expression (value or quantity).

**Type**

Object

**Parameter**

DataReference of the parameter.

**Type**

DataReference
Example

The following example shows how to set the expression for one parameter in an existing custom candidate point.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
customCandidatePoint = model.GetCustomCandidatePoint(Name="CustomCandidatePoint 1")
inputParameter1 = model.GetParameter(Name="P1")
customCandidatePoint.SetParameter(Parameter=inputParameter1, Expression="2.01")
inputParameter2 = model.GetParameter(Name="P2")
customCandidatePoint.SetParameter(Parameter=inputParameter2, Expression="2.1 [m]")
```

**SetParameters**

Sets the expression of several parameters in a CustomCandidatePoint entity. In the context of a response surface based optimization, the output values are updated from the response surface.

**Required Arguments**

**Expressions**  
Dictionary of the parameters and their assigned expressions.

**Type**  
IDictionary<DataReference, Object>

**Example**

The following example shows how to set the expression for several parameters in an existing custom candidate point.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
customCandidatePoint = model.GetCustomCandidatePoint(Name="CustomCandidatePoint 1")
inputParameter1 = model.GetParameter(Name="P1")
customCandidatePoint.SetParameter(Parameter=inputParameter1, Expression="2.01")
inputParameter2 = model.GetParameter(Name="P2")
customCandidatePoint.SetParameters(Expressions={inputParameter1: "2.01", inputParameter2: "15.5 [m]"})
```

**Update**

Updates a point entity to provide the output parameter values of the requested source. If the requested source is ResponseSurface, the output values are generated by evaluating the response surface model, if available. If the requested source is Simulation, the output values are generated by triggering a DesignPoint update. This command applies to CandidatePoint and CustomCandidatePoint entities.

**Return**

The dictionary of parameters with their values.

**Type**  
ReadOnlyDictionary<DataReference, Object>

**Optional Arguments**

**Source**  
Source of output values to produce. The update method depends on this parameter.

**Type**  
OutputSource

**Default Value**  
Simulation
Example
The following example shows how to update an existing candidate point to obtain Simulation output values.

```python
system1 = GetSystem(Name="RSO")
container = system1.GetContainer(ComponentName="Optimization")
candidate1 = container.GetCandidatePoint(Name="CandidatePoint 1")
outputValues = candidate1.Update(Source="Simulation")
```

**DesignPointsCurvesChart**

The data entity which describes a "Design Points vs. Parameters" chart. It provides a 2D chart with two Y axes and two X axes to display Design Points versus Parameter and/or Parameter versus Parameter curves.

**Properties**

**Axes**

Dictionary of the parameters associated to axes

**Type** Dictionary<ChartAxes, DataReference>

**Read Only** Yes

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

**Type** bool

**Read Only** No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

**Read Only** No

**IsUpToDate**

True if the entity is up-to-date.

**Type** bool

**Read Only** No
Methods

AssociateParameterToAxis

Associates a Parameter to the specified axis of the chart. The Parameter argument can be omitted which means that the axis is not set.

Required Arguments

Axis  
Axis to modify.

Type  
ChartAxes

Optional Arguments

Parameter  
Parameter entity to be assigned to the specified axis.

Type  
DataReference

Example

The following example shows how to assign parameters to the axes of a DesignPointsCurves chart.

```
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
inputParam1 = model.GetParam(Name="P1")
outputParam1 = model.GetParam(Name="P5")
outputParam2 = model.GetParam(Name="P6")
chart = model.GetChart(Name="Design Point vs Parameter 1")
chart.AssociateParameterToAxis(Parameter=inputParam1, Axis="XAxis")
chart.AssociateParameterToAxis(Parameter=outputParam1, Axis="YAxis")
chart.AssociateParameterToAxis(Parameter=outputParam2, Axis="YRightAxis")
```

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName  
The exported file name.

Type  
string

Optional Arguments

AppendMode  
True to append to an existing csv file, False to overwrite it.

Type  
bool

Default Value  
False

Example

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```
container = system1.GetContainer(ComponentName="Design of Experiment")
```
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)

**Update**

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

**Example**

The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

**DesignPointsParallelChart**

The data entity which describes a Parameters Parallel chart. It allows you to visualize the DOE matrix using parallel Y axes to represent all of the input and output parameters on the same 2D representation, whatever the number of parameters.

**Properties**

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

- **Type** bool
- **Read Only** No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type** string
- **Read Only** No

**IsUpToDate**

True if the entity is up-to-date.

- **Type** bool
- **Read Only** No
Methods

EnableParameters

Enable or disable a list of parameters in a chart.

Required Arguments

IsEnabled  
False to disable the parameters, or True (default) to enable them.

Type  bool

Optional Arguments

Parameters  
Parameters to enable or disable, or all parameters if not specified.

Type  List<DataReference>

Example

The following example shows how to disable two parameters, and then how to enable all parameters by omitting the Parameters optional parameter. This example uses a Correlation Matrix chart. The method also applies for other types of charts like LocalSensitivity, SpiderChart, etc.

```python
container = system1.GetContainer(ComponentName="Correlation")
model = container.GetModel()
param1 = model.GetParam(Name="P1")
param4 = model.GetParam(Name="P4")
chart = model.GetChart(Name="Correlation Matrix 1")
chart.EnableParameters(Parameters=[param1;param4], IsEnabled=false)
chart.EnableParameters()
```

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName  
The exported file name.

Type  string

Optional Arguments

AppendMode  
True to append to an existing csv file, False to overwrite it.

Type  bool

Default Value  False

Example

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
```
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)

**Update**

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

**Example**

The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

**DeterminationHistogramChart**

The data entity which describes the Determination Histogram chart. It allows you to visualize the coefficient of determination (linear or quadratic) of each input for an output parameter.

**Properties**

**Axes**

Dictionary of the parameters associated to axes.

- **Type** Dictionary<ChartAxes, DataReference>
- **Read Only** Yes

**DeterminationType**

Determination type, either Linear or Quadratic.

- **Type** DeterminationCoefficientChartModes
- **Read Only** No

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

- **Type** bool
- **Read Only** No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.
**Type**  
string

**Read Only**  
No

**FullModelR2**

Full Model R2 value.

**Type**  
double

**Read Only**  
Yes

**IsUpToDate**

True if the entity is up-to-date.

**Type**  
bool

**Read Only**  
No

**ThresholdR2**

Threshold value of R2: the chart displays only input parameters with a coefficient of determination is greater than thus threshold.

**Type**  
double

**Read Only**  
No

**Methods**

**AssociateParameterToAxis**

Associates a Parameter to the specified axis of the chart. The Parameter argument can be omitted which means that the axis is not set.

**Required Arguments**

**Axis**  
Axis to modify.

**Type**  
ChartAxes

**Optional Arguments**

**Parameter**  
Parameter entity to be assigned to the specified axis.

**Type**  
DataReference

**Example**

The following example shows how to assign parameters to the axes of a DesignPointsCurves chart.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
inputParam1 = model.GetParam(Name="P1")
outputParam1 = model.GetParam(Name="P5")
outputParam2 = model.GetParam(Name="P6")
chart = model.GetChart(Name="Design Point vs Parameter 1")
```
chart.AssociateParameterToAxis(Parameter=inputParam1, Axis="XAxis")
chart.AssociateParameterToAxis(Parameter=outputParam1, Axis="YAxis")
chart.AssociateParameterToAxis(Parameter=outputParam2, Axis="YRightAxis")

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName  The exported file name.

Type  string

Optional Arguments

AppendMode  True to append to an existing csv file, False to overwrite it.

Type  bool

Default Value  False

Example

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

Update

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

Example

The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

DeterminationMatrixChart

The data entity which describes a Determination Matrix chart. This matrix allows you to visualize how the input and output parameters are coupled in a quadratic regression.
**Properties**

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

- **Type**: bool
- **Read Only**: No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**IsUpToDate**

True if the entity is up-to-date.

- **Type**: bool
- **Read Only**: No

**Methods**

**EnableParameters**

Enable or disable a list of parameters in a chart.

**Required Arguments**

**IsEnabled**

False to disable the parameters, or True (default) to enable them.

- **Type**: bool

**Optional Arguments**

**Parameters**

Parameters to enable or disable, or all parameters if not specified.

- **Type**: List<DataReference>

**Example**

The following example shows how to disable two parameters, and then how to enable all parameters by omitting the Parameters optional parameter. This example uses a Correlation Matrix chart. The method also applies for other types of charts like LocalSensitivity, SpiderChart, etc.

```python
container = system1.GetContainer(ComponentName="Correlation")
model = container.GetModel()
param1 = model.GetParam(Name="P1")
param4 = model.GetParam(Name="P4")
chart = model.GetChart(Name="Correlation Matrix 1")
```

- **Release 15.0 - © SAS IP Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.**
chart.EnableParameters( Parameters=[param1,param4], IsEnabled=false )
chart.EnableParameters()

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName  The exported file name.

  Type   string

Optional Arguments

AppendMode  True to append to an existing csv file, False to overwrite it.

  Type   bool

  Default Value   False

Example

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

Update

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

Example

The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

DistributionChart

The entity which describes a parameter's Distribution chart. This chart is always associated with an uncertainty parameter and allows you to visualize the statistical distribution defined or calculated for this parameter.
Properties

DisplayParameterFullName

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

Type: bool
Read Only: No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string
Read Only: No

IsUpToDate

True if the entity is up-to-date.

Type: bool
Read Only: No

Parameter

The parameter entity associated with the chart.

Type: DataReference
Read Only: No

Methods

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName: The exported file name.

Type: string

Optional Arguments

AppendMode: True to append to an existing csv file, False to overwrite it.

Type: bool
Default Value: False
Example
The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

Update
Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

Example
The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

GoodnessOfFit
Entity which manages the Goodness Of Fit Information of a Response Surface for an Output Parameter

Properties

DiscreteExpressions
A Dictionary holding the values of all discrete input. Note: Discrete parameter values are level values, not indices.

Type: Dictionary<DataReference, Object>

Read Only: No

DisplayText
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string

Read Only: No

IsUpToDate
True if the entity is up-to-date.

Type: bool
**Methods**

**CreateAdvancedReport**

For the standard response surface only (Full second order Polynomials), creates an Advanced Goodness of Fit report for any direct output parameter.

**Return**

A string which contains the generated Advanced Goodness of Fit report.

**Required Arguments**

**Parameter**

Parent Parameter of the report.

**Type** DataReference

**Optional Arguments**

**PlainTextFormat**

Plain text formatting instead of HTML (default).

**Type** bool

**Default Value** False

**Example**

The following example shows how to create an Advanced Goodness of Fit report.

```python
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
go1 = model.GetGoodnessOfFit(Name="GoodnessOfFit")
outputParameter1 = model.GetParameter(Name="P4")
report1 = gof1.CreateAdvancedReport(Parameter=outputParameter1)
```

**SetDiscreteParameter**

Sets the Expression of a discrete input parameter in a GoodnessOfFit and updates the associated output parameter values. The chart entities depending on the GoodnessOfFit are updated as well.

**Required Arguments**

**DiscreteParameter**

DataReference of the Discrete Input parameter.

**Type** DataReference

**Expression**

Assigned Expression (discrete value).

**Type** string

**Example**

The following example shows how to set the expression for one discrete parameter in an existing GoodnessOfFit.
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
gof = model.GetGoodnessOfFit(Name="Goodness Of Fit")
inputParameter1 = model.GetParameter(Name="P1")
gof.SetDiscreteParameter(DiscreteParameter=inputParameter1, Expression="15")

**Update**

Updates the goodness of fit and the results which depend on it. If the goodness of fit is already up-to-date, nothing is done unless the Force flag is set to True.

**Optional Arguments**

**Force**  
Set to true if the update operation of the goodness of fit point is required even if it's already up-to-date.

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>False</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how to update an existing GoodnessOfFit.

```python
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
gof1 = model.GetGoodnessOfFit(Name="GOF 1")
gof1.Update()
```

**HistoryChart**

The data entity which describes an History chart. It allows you to visualize how a parameter evolves during the optimization process, point by point or iteration by iteration depending on the optimizer. Defined target and constraints values are also represented.

**Properties**

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>
**IsUpToDate**

True if the entity is up-to-date.

**Type**  
bool

**Read Only**  
No

**Methods**

**ExportData**

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

**Required Arguments**

**FileName**  
The exported file name.

**Type**  
string

**Optional Arguments**

**AppendMode**  
True to append to an existing csv file, False to overwrite it.

**Type**  
bool

**Default Value**  
False

**Example**

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

**Update**

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

**Example**

The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```
**LocalSensitivityChart**

The data entity which describes a Local Sensitivity chart. It allows you to visualize the impact that changing each input parameter independently has on the output parameters. A Local Sensitivity chart accepts two different graphical modes - BarChart and PieChart - and supports enabling/disabling input and/or output parameters. A Local Sensitivity depends on a Response Point which serves as a reference point for sensitivity computation.

**Properties**

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

*Type* `bool`

*Read Only* No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type* `string`

*Read Only* No

**IsUpToDate**

True if the entity is up-to-date.

*Type* `bool`

*Read Only* No

**Mode**

Specifies the graphical representation used to render local sensitivity data. It can be either BarChart or PieChart.

*Type* `SensitivityChartModes`

*Read Only* No

**Methods**

**EnableParameters**

Enable or disable a list of parameters in a chart.

**Required Arguments**

**IsEnabled** False to disable the parameters, or True (default) to enable them.
**Optional Arguments**

**Parameters**  Parameters to enable or disable, or all parameters if not specified.

**Type**  List<DataReference>

**Example**
The following example shows how to disable two parameters, and then how to enable all parameters by omitting the Parameters optional parameter. This example uses a Correlation Matrix chart. The method also applies for other types of charts like LocalSensitivity, SpiderChart, etc.

```python
container = system1.GetContainer(ComponentName="Correlation")
model = container.GetModel()
param1 = model.GetParam(Name="P1")
param4 = model.GetParam(Name="P4")
chart = model.GetChart(Name="Correlation Matrix 1")
chart.EnableParameters(Parameters=[param1,param4], IsEnabled=false)
chart.EnableParameters()
```

**ExportData**

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

**Required Arguments**

**FileName**  The exported file name.

**Type**  string

**Optional Arguments**

**AppendMode**  True to append to an existing csv file, False to overwrite it.

**Type**  bool

**Default Value**  False

**Example**
The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

**Update**

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.
Example
The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```python
ccontainer = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

**LocalSensitivityCurvesChart**

The data entity which describes a Local Sensitivity Curves chart. It allows you to visualize the impact that changing each input parameter independently has on the output parameters. It allows you to visualize one or two output parameters at the same time. A Local Sensitivity Curves chart supports enabling and disabling input parameters. A Local Sensitivity Curves depends on a Response Point which serves as a reference point for computation of the sensitivity.

**Properties**

**Axes**

Dictionary of the parameters associated to axes.

**Type** Dictionary<ChartAxes, DataReference>

**Read Only** Yes

**AxesRangeMode**

Axes range for output parameters controls if the range is determined from the chart's data or the min-max of outputs.

**Type** AxesRangeModes

**Read Only** No

**ChartResolution**

Chart resolution.

**Type** int

**Read Only** No

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

**Type** bool

**Read Only** No
**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

**IsUpToDate**

True if the entity is up-to-date.

**Type**  
bool

**Read Only**  
No

**Methods**

**AssociateParameterToAxis**

Associates a Parameter to the specified axis of the chart. The Parameter argument can be omitted which means that the axis is not set.

**Required Arguments**

**Axis**  
Axis to modify.

**Type**  
ChartAxes

**Optional Arguments**

**Parameter**  
Parameter entity to be assigned to the specified axis.

**Type**  
DataReference

**Example**

The following example shows how to assign parameters to the axes of a DesignPointsCurves chart.

```plaintext
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
inputParam1 = model.GetParam(Name="P1")
outputParam1 = model.GetParam(Name="P5")
outputParam2 = model.GetParam(Name="P6")
chart = model.GetChart(Name="Design Point vs Parameter 1")
chart.AssociateParameterToAxis(Parameter=inputParam1, Axis="XAxis")
chart.AssociateParameterToAxis(Parameter=outputParam1, Axis="YAxis")
chart.AssociateParameterToAxis(Parameter=outputParam2, Axis="YRightAxis")
```

**EnableParameters**

Enable or disable a list of parameters in a chart.

**Required Arguments**

**IsEnabled**  
False to disable the parameters, or True (default) to enable them.
**Type**  
bool

**Optional Arguments**

**Parameters**  
Parameters to enable or disable, or all parameters if not specified.

**Type**  
List<DataReference>

**Example**  
The following example shows how to disable two parameters, and then how to enable all parameters by omitting the Parameters optional parameter. This example uses a Correlation Matrix chart. The method also applies for other types of charts like LocalSensitivity, SpiderChart, etc.

```python
container = system1.GetContainer(ComponentName="Correlation")
model = container.GetModel()
param1 = model.GetParam(Name="P1")
param4 = model.GetParam(Name="P4")
chart = model.GetChart(Name="Correlation Matrix 1")
chart.EnableParameters(Parameters=[param1;param4], IsEnabled=false)
chart.EnableParameters()
```

**ExportData**

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

**Required Arguments**

**FileName**  
The exported file name.

**Type**  
string

**Optional Arguments**

**AppendMode**  
True to append to an existing csv file, False to overwrite it.

**Type**  
bool

**Default Value**  
False

**Example**  
The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

**Update**

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.
Example

The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

**ParameterStatistics**

Entity which manages the calculated Statistics of a parameter for a Six-Sigma Component

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**Entropy**

Shannon Entropy value (Complexity)

- **Type**: double
- **Read Only**: No

**InvProbabilityTable**

DataReference of the Inverse Probability ParametricTable for the associated parameter

- **Type**: DataReference
- **Read Only**: No

**IsUpToDate**

True if the entity is up-to-date.

- **Type**: bool
- **Read Only**: No

**Kurtosis**

Kurtosis value

- **Type**: double
- **Read Only**: No
**Mean**

Mean value

**Type**  double

**Read Only**  No

**Parameter**

DataReference of the input or output parameter associated with this statistics entity.

**Type**  DataReference

**Read Only**  No

**ProbabilityTable**

DataReference of the Probability ParametricTable for the associated parameter

**Type**  DataReference

**Read Only**  No

**SigmaMaximum**

Sigma Maximum value

**Type**  double

**Read Only**  No

**SigmaMinimum**

Sigma Minimum value

**Type**  double

**Read Only**  No

**SignalNoiseLarge**

Signal-Noise Ratio value (Larger is Better)

**Type**  double

**Read Only**  No

**SignalNoiseNominal**

Signal-Noise Ratio value (Nominal is Best)

**Type**  double

**Read Only**  No
**SignalNoiseSmall**

Signal-Noise Ratio value (Smaller is Better)

**Type**  
double

**Read Only**  
No

**Skewness**

Skewness value

**Type**  
double

**Read Only**  
No

**StandardDeviation**

Standard Deviation value

**Type**  
double

**Read Only**  
No

**StatisticsChart**

DataReference of the statistics chart for the associated parameter

**Type**  
DataReference

**Read Only**  
No

**TableType**

Type of the probability table

**Type**  
SixSigmaTableTypes

**Read Only**  
No

**ParametricTable**

ParametricTable entity used to encapsulate most of the evaluation results in a convenient 2D matrix format.

**Properties**

**DimCol**

Number of columns.

**Type**  
int

**Read Only**  
Yes
**DimRow**

Number of rows.

**Type**  int

**Read Only**  Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  string

**Read Only**  No

**Methods**

**AddRow**

Adds a row to the bottom of a ParametricTable entity.

**Optional Arguments**

**RowValues**  New values for the row.

**Type**  List<string>

**Example**

The following example shows how to make a DOE editable, to retrieve the table of design points and add a new row to it.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
model.DOETYPE = "eDOETYPE_USER"
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.AddRow()
```

**DeleteRows**

Delete rows from a ParametricTable entity.

**Required Arguments**

**Indices**  Indices of the rows to delete.

**Type**  List<int>

**Example**

The following example shows how to delete rows from the DOEmatrix in a custom DOE context.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
```
DOEMatrix.DeleteRows(Indices=[0,7,8,9])

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName    The exported file name.

Type      string

Optional Arguments

AppendMode  True to append to an existing csv file, False to overwrite it.

Type      bool

Default Value  False

Example

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)

GetCellValue

Get the Value of a ParametricTable's cell. An exception is thrown if the entity is not found.

Return  The value of the cell.

Type      string

Required Arguments

ColumnIndex  ColumnIndex (zero-based) of the cell.

Type      int

RowIndex  RowIndex (zero-based) of the cell.

Type      int

Example

The following example shows how the user can get the value of the ParametricTable's cell [0,3].

system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModell = designofExperiment1.GetModel()
parametricTable = dOEModel1.GetParametricTable(Name="DesignPoints")
cellValue = parametricTable.GetCellValue(RowIndex=0,ColumnIndex=3)

**GetRowUpdateOrder**

Get the Update Order value of a row in a ParametricTable. An exception is thrown if the entity is not found.

**Return**

The value of the update order.

**Type**

double

**Required Arguments**

**RowIndex**

RowIndex (zero-based) of the cell.

**Type**

int

**Example**

The following example shows how to get the Update Order value of row [3] in the ParametricTable.

```csharp
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
parametricTable = dOEModel1.GetParametricTable(Name="DesignPoints")
updateOrderValue = parametricTable.GetRowUpdateOrder(RowIndex=3)
```

**GetRowValues**

Get the Values of a ParametricTable's row. An exception is thrown if the entity is not found.

**Return**

List of the values of the specified row.

**Type**

List<string>

**Required Arguments**

**RowIndex**

RowIndex (zero-based) of the row to retrieve.

**Type**

int

**Example**

The following example shows how the user can get the values of the 4th ParametricTable's row.

```csharp
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
parametricTable = dOEModel1.GetParametricTable(Name="DesignPoints")
rowValues = parametricTable.GetRowValues(RowIndex=3)
```

**OptimizeUpdateOrder**

Optimizes the Update Order of Design Points to minimize the number of modifications between two consecutive Design Points. This command applies to the "DesignPoints" ParametricTable of a Design
of Experiments model or of a Response Surface in a manual refinement context. It also applies to the "VerificationPoints" ParametricTable of a Response Surface model.

**Example**

The following example shows how to optimize the update order of the DesignPoints table in a Design of Experiments model.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
model.DOETYPE = "eDOETYPE_USER"
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.OptimizeUpdateOrder()
```

**SetCellValue**

Sets the value of a ParametricTable cell. If the table is read-only, the command has no effect.

**Required Arguments**

- **ColumnIndex** Zero-based column index of the cell.
  - Type: int
- **RowIndex** Zero-based row index of the cell.
  - Type: int
- **Value** New value of the cell.
  - Type: string

**Example**

The following example shows how to set the value of the DesignPoints table in a custom DOE context.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
model.DOETYPE = "eDOETYPE_USER"
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.AddRow()
DOEMatrix.SetCellValue(RowIndex=0, ColumnIndex=0, Value="12.5 [mm]")
```

**SetOutputValuesEditable**

Sets the values of the output parameters as Editable (Editable=True) or Calculated (Editable=False) for the complete table, or a set of rows specified by the RowIndices argument. This command is applicable to the "DesignPoints" ParametricTable of a Design of Experiments model in a custom context (DOETYPE is "eDOETYPE_USER" or "eDOETYPE_CUSTOM_OSF") or of a Response Surface model in a manual refinement context. It also applies to the "VerificationPoints" ParametricTable of a Response Surface model.

**Required Arguments**

- **Editable** True to define output values as Editable, or False to mark output values as calculated.
  - Type: bool
Optional Arguments

**RowIndices** Optional list of row zero-based indices. If this argument is not specified, the command applies to the complete ParametricTable.

  **Type** List<int>

Example
The following example shows how to switch a DOE model to the custom mode and then set the output values as editable for the first three design points of the DOE matrix.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
model.DOETYPE = "eDOETYPE_USER"
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.SetOutputValuesEditable(Editable=True, RowIndices=[0,1,2])
```

**SetRowUpdateOrder**

Sets the Update Order value for a row in a ParametricTable. If the table doesn't support Update Order functionality, the command has no effect.

**Required Arguments**

**RowIndex** Zero-based row index of the cell.

  **Type** int

**UpdateOrder** New value of the update order.

  **Type** double

Example
The following example shows how to set the Update Order value of the DesignPoints table in a Design of Experiments model.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.SetRowUpdateOrder(RowIndex=0, Value="2.0")
```

**SetRowValues**

Sets the values of a ParametricTable's row. If the table is read-only, the command has no effect.

**Required Arguments**

**RowIndex** Zero-based row index of the cell.

  **Type** int

**RowValues** New values for the row.

  **Type** List<string>
Example
The following example shows how to set the values of the DesignPoints table in a custom DOE context.

```
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
model.DOEType = "eDOETYPE_USER"
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.AddRow()
DOEMatrix.SetRowValues(RowIndex=0, RowValues=[ "12.5 [mm]", "1" ] )
```

```
SetUpdateOrderByRow

Sets a value for the UpdateOrder property of all Design Points using sorting settings. This command applies to the "DesignPoints" ParametricTable of a Design of Experiments model or of a Response Surface in a manual refinement context. It also applies to the "VerificationPoints" ParametricTable of a Response Surface model.

Optional Arguments

SortBy Definition of the sort.

Type List<string>

Example
The following example shows how to set the update order of the DesignPoints table in a Design of Experiments model. The Design Points are sorted first by their values for the parameter P3 (in ascending order), and then by their values for the parameter P1 (in descending order).

```
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.SetUpdateOrderByRow(SortBy=["P3:+", "P1:-" ])
```

```
UpdateRows

Updates the design points held in rows from a ParametricTable entity and the results which depend on it.

Required Arguments

Indices Indices of the rows to update.

Type List<int>

Example
The following example shows how to update the design points from the VerificationPoints Table.

```
responseSurface1 = system1.GetContainer(ComponentName="Response Surface")
responseSurfaceModel1 = responseSurface1.GetModel()
parametricTable1 = responseSurfaceModel1.GetParametricTable(Name="VerificationPoints")
parametricTable1.UpdateRows(Indices=[0,1])
```
**PredictedvsObservedScatterChart**

The data entity which describes a PredictedvsObserved Scatter chart. This scatter chart allows the user to visualize the predicted output values versus observed output values for all design points used to compute the response surface.

**Properties**

*DisplayParameterFullName*

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

**Type** bool

**Read Only** No

*DisplayText*

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

**Read Only** No

*IsUpToDate*

True if the entity is up-to-date.

**Type** bool

**Read Only** No

*ShowDesignPoints*

Option to display the design points.

**Type** bool

**Read Only** No

*ShowVerificationPoints*

Option to display the verification points.

**Type** bool

**Read Only** No

**Methods**

*EnableParameters*

Enable or disable a list of parameters in a chart.
Required Arguments

IsEnabled  False to disable the parameters, or True (default) to enable them.

Type    bool

Optional Arguments

Parameters  Parameters to enable or disable, or all parameters if not specified.

Type    List<DataReference>

Example

The following example shows how to disable two parameters, and then how to enable all parameters by omitting the Parameters optional parameter. This example uses a Correlation Matrix chart. The method also applies for other types of charts like LocalSensitivity, SpiderChart, etc.

```csharp
container = system1.GetContainer(ComponentName="Correlation")
model = container.GetModel()
param1 = model.GetParam(Name="P1")
param4 = model.GetParam(Name="P4")
chart = model.GetChart(Name="Correlation Matrix 1")
chart.EnableParameters(Parameters=[param1;param4], IsEnabled=false)
chart.EnableParameters()
```

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName  The exported file name.

Type    string

Optional Arguments

AppendMode  True to append to an existing csv file, False to overwrite it.

Type    bool

Default Value    False

Example

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```csharp
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```
**Update**

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

**Example**

The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```plaintext
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

**ResponseChart**

The data entity which describes a Response chart. It allows you to visualize the impact that changing each input parameter has on the selected output parameter. It has three Modes - 2D, 3D and 2DSlices - allowing you to vary one or two input parameters at the same time. A Response depends on a Response Point which serves as a reference point for non-varying input parameters.

**Properties**

**Axes**

Dictionary of the parameters associated to axes

**Type**  
Dictionary<ChartAxes, DataReference>

**Read Only**  
Yes

**ChartResolutionAlongX**

Chart resolution along the X axis.

**Type**  
int

**Read Only**  
No

**ChartResolutionAlongY**

Chart resolution along the Y axis.

**Type**  
int

**Read Only**  
No

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

**Type**  
bool

**Read Only**  
No
**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

**Read Only** No

**IsUpToDate**

True if the entity is up-to-date.

**Type** bool

**Read Only** No

**Mode**

Response chart mode: 2D, 3D or 2D Slices.

**Type** ResponseChartModes

**Read Only** No

**NumberOfSlices**

Number of slices for the 2D Slices mode.

**Type** int

**Read Only** No

**ShowDesignPoints**

Option to display the design points of the DOE and the refinement points.

**Type** bool

**Read Only** No

**Methods**

**AssociateParameterToAxis**

Associates a Parameter to the specified axis of the chart. The Parameter argument can be omitted which means that the axis is not set.

**Required Arguments**

**Axis** Axis to modify.

**Type** ChartAxes
Optional Arguments

**Parameter** Parameter entity to be assigned to the specified axis.

  **Type** DataReference

**Example**
The following example shows how to assign parameters to the axes of a DesignPointsCurves chart.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
inputParam1 = model.GetParam(Name="P1")
outputParam1 = model.GetParam(Name="P5")
outputParam2 = model.GetParam(Name="P6")
chart = model.GetChart(Name="Design Point vs Parameter 1")
chart.AssociateParameterToAxis(Parameter=inputParam1, Axis="XAxis")
chart.AssociateParameterToAxis(Parameter=outputParam1, Axis="YAxis")
chart.AssociateParameterToAxis(Parameter=outputParam2, Axis="YRightAxis")
```

**ExportData**

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

**Required Arguments**

**FileName** The exported file name.

  **Type** string

**Optional Arguments**

**AppendMode** True to append to an existing csv file, False to overwrite it.

  **Type** bool

  **Default Value** False

**Example**
The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

**Update**

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.
Example
The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

**ResponsePoint**

The ResponsePoint data entity: a design point for which output values are computed from a Response Surface.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**Expressions**

A Dictionary of the input parameter values defining the response point and the corresponding output parameter values predicted from the response surface. Note that for discrete parameters and continuous parameters using manufacturable values, the Dictionary contains the value of the level rather than its index.

- **Type**: Dictionary<
  - DataReference,
  - Object>
- **Read Only**: No

**Note**

User notes about the response point.

- **Type**: string
- **Read Only**: No

**Methods**

**CreateChart**

Creates a Chart entity attached to the Response Point. This chart is updated automatically when the parameter values of the ResponsePoint change. It is deleted automatically when the ResponsePoint is deleted. The chart must be updated once after its creation by invoking its Update method. Then changes to its properties or its parent ResponsePoint will update the chart automatically.

- **Return**: The DataReference of the Chart.
**DataReference**

**Type**  
DataReference

**Required Arguments**

**ChartType**  
Type of chart to be created. The possible values are "eChartLocalSensitivity", "eChartLocalSensitivityCurves", "eChartResponse" and "eChartSpiderResponses"

<table>
<thead>
<tr>
<th>Type</th>
<th>ChartType</th>
</tr>
</thead>
</table>

**Optional Arguments**

**DisplayText**  
DisplayText of the chart. If not specified, a default name is applied, depending on the value of ChartType.

| Type | string |

**Example**

The following example shows how to create a ResponsePoint chart.

```python
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
responsePoint = model.GetResponsePoint(Name="Response Point 1")
spiderChart = responsePoint.CreateChart(ChartType="eChartSpiderResponses")
spiderChart.Update()
```

**DuplicateChart**

Duplicates a Chart entity attached to a Response Point. The chart must be updated once after its creation by invoking its Update method. Then changes to its properties will update the chart automatically.

**Return**

The DataReference of the Chart.

| Type | DataReference |

**Required Arguments**

**Chart**  
The source chart to duplicate.

| Type | DataReference |

**Optional Arguments**

**DisplayText**  
Displayed name of the chart. If not specified, a default name is applied, depending on the value of ChartType.

| Type | string |

**TargetResponsePoint**  
ResponsePoint on which the duplicated chart is created. If this parameter is not set, the response point of the source chart is used.

| Type | DataReference |

**Example**

The following example shows how to duplicate a chart.
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
responsePoint = model.GetResponsePoint(Name="Response Point 1")
spiderChart = responsePoint.GetChart(Name="Spider")
spiderChart2 = responsePoint.DuplicateChart(Chart=spiderChart)
spiderChart2.Update()
**Type**  
*Dictionary<DataReference, string>*

**Example**  
The following example shows how to set the expression for several discrete input parameters in an existing GoodnessOfFit.

```plaintext
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
gof = model.GetGoodnessOfFit(Name="Goodness Of Fit 1")
inputParameter1 = model.GetParameter(Name="P1")
inputParameter2 = model.GetParameter(Name="P2")
gof.SetDiscreteParameters(Expressions={inputParameter1: "2", inputParameter2: "15"})
```

**SetParameter**

Sets the Expression of an input parameter in a ResponsePoint and updates the associated output parameter values. If there are chart entities depending on the ResponsePoint, they are updated as well.

**Required Arguments**

**Expression**  
Assigned Expression (value or quantity).

- **Type**  
*string*

**Parameter**  
DataReference of the Input parameter.

- **Type**  
*DataReference*

**Example**  
The following example shows how to set the expression for one parameter in an existing ResponsePoint.

```plaintext
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
 rp = model.GetResponsePoint(Name="Response Point 1")
inputParameter1 = model.GetParameter(Name="P1")
rp.SetParameter(Parameter=inputParameter1, Expression="2.01")
inputParameter2 = model.GetParameter(Name="P2")
rp.SetParameter(Parameter=inputParameter2,Expression="2.1 [m]")
```

**SetParameters**

Sets the Expression of several input parameters in a ResponsePoint and updates the associated output parameter values. If there are chart entities depending on the ResponsePoint, they are updated as well.

**Required Arguments**

**Expressions**  
Dictionary of the input parameters and their assigned expressions.

- **Type**  
*Dictionary<DataReference, string>*

**Example**  
The following example shows how to set the expression for several input parameters in an existing ResponsePoint.

```plaintext
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
```
rp = model.GetResponsePoint(Name="Response Point 1")
inputParameter1 = model.GetParameter(Name="P1")
inputParameter2 = model.GetParameter(Name="P2")
rp.SetParameters(Expressions={inputParameter1: "2.01", inputParameter2: "15.5 [m]"})

**Update**

Updates the response point and the results which depend on it. If the response point is already up to date, nothing is done unless the Force flag is set to True.

**Optional Arguments**

**Force**

Set to true if the update operation of the response point is required even if it's already up to date.

- **Type**: bool
- **Default Value**: False

**Example**

The following example shows how to update an existing ResponsePoint.

```python
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
rp = model.GetResponsePoint(Name="Response Point 1")
rp.Update()
```

**SamplesChart**

The data entity which describes the Samples chart. It allows you to explore the samples generated for an Optimization study by using parallel Y axes to represent all of the input and output parameters on the same 2D representation, whatever the number of parameters. It can filter the number of visible Pareto Fronts and supports two Modes: Candidates or Pareto Fronts.

**Properties**

**ColoringMethod**

Coloring method used to draw the chart.

- **Type**: ChartColoringMethods
- **Read Only**: No

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

- **Type**: bool
- **Read Only**: No
**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** `string`
**Read Only** No

**IsUpToDate**

True if the entity is up-to-date.

**Type** `bool`
**Read Only** No

**Mode**

Samples chart mode.

**Type** `SamplesChartModes`
**Read Only** No

**NumberOfParetoFront**

Number of Pareto fronts to display. This is used as a filter to display only the most interesting fronts, given an optimization study.

**Type** `uint`
**Read Only** No

**ShowInfeasiblePoints**

If True, any infeasible points are displayed on the chart.

**Type** `bool`
**Read Only** No

**Methods**

**EnableParameters**

Enable or disable a list of parameters in a chart.

**Required Arguments**

**IsEnabled**

False to disable the parameters, or True (default) to enable them.

**Type** `bool`
Optional Arguments

Parameters  Parameters to enable or disable, or all parameters if not specified.

  Type  List<DataReference>

Example
The following example shows how to disable two parameters, and then how to enable all parameters by omitting the Parameters optional parameter. This example uses a Correlation Matrix chart. The method also applies for other types of charts like LocalSensitivity, SpiderChart, etc.

```csharp
container = system1.GetContainer(ComponentName="Correlation")
model = container.GetModel()
param1 = model.GetParam(Name="P1")
param4 = model.GetParam(Name="P4")
chart = model.GetChart(Name="Correlation Matrix 1")
chart.EnableParameters(Parameters=[param1;param4], IsEnabled=false )
chart.EnableParameters()
```

ExportData
Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName  The exported file name.

  Type  string

Optional Arguments

AppendMode  True to append to an existing csv file, False to overwrite it.

  Type  bool

  Default Value  False

Example
The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```csharp
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

Update
Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.
Example
The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```
class = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

**SensitivitiesChart**

The data entity which describes a Sensitivities chart. It allows you to visualize the global sensitivities of each output with respect to the input parameters. A Sensitivities chart has two different graphical modes - BarChart and PieChart- and supports enabling/disabling input and/or output parameters.

**Properties**

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**IsUpToDate**

True if the entity is up-to-date.

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**Mode**

Specifies the graphical representation used to render local sensitivity data. It can be either BarChart or PieChart.

<table>
<thead>
<tr>
<th>Type</th>
<th>SensitivityChartModes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>
Methods

EnableParameters

Enable or disable a list of parameters in a chart.

Required Arguments

isEnabled  False to disable the parameters, or True (default) to enable them.

Type  bool

Optional Arguments

Parameters  Parameters to enable or disable, or all parameters if not specified.

Type  List<DataReference>

Example

The following example shows how to disable two parameters, and then how to enable all parameters by omitting the Parameters optional parameter. This example uses a Correlation Matrix chart. The method also applies for other types of charts like LocalSensitivity, SpiderChart, etc.

```java
container = system1.GetContainer(ComponentName="Correlation")
model = container.GetModel()
param1 = model.GetParam(Name="P1")
param4 = model.GetParam(Name="P4")
chart = model.GetChart(Name="Correlation Matrix 1")
chart.EnableParameters(Parameters=[param1,param4], isEnabled=false)
chart.EnableParameters()
```

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName  The exported file name.

Type  string

Optional Arguments

AppendMode  True to append to an existing csv file, False to overwrite it.

Type  bool

Default Value  False

Example

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```java
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
```
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParalell")
chart.ExportData(FileName="doe.csv", AppendMode=True)

**Update**

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

**Example**
The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

**SpiderChart**

The data entity which describes a Spider chart. It allows you to visualize the impact that changing the input parameters has on all output parameters simultaneously. A Spider chart depends on a Response Point and shows the same values as its parent Response Point.

**Properties**

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

*Type*  
bool  
*Read Only*  
No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type*  
string  
*Read Only*  
No

**IsUpToDate**

True if the entity is up-to-date.

*Type*  
bool  
*Read Only*  
No
Methods

EnableParameters

Enable or disable a list of parameters in a chart.

Required Arguments

IsEnabled  False to disable the parameters, or True (default) to enable them.

    Type       bool

Optional Arguments

Parameters  Parameters to enable or disable, or all parameters if not specified.

    Type       List<DataReference>

Example

The following example shows how to disable two parameters, and then how to enable all parameters by omitting the Parameters optional parameter. This example uses a Correlation Matrix chart. The method also applies for other types of charts like LocalSensitivity, SpiderChart, etc.

    container = system1.GetContainer(ComponentName="Correlation")
    model = container.GetModel()
    param1 = model.GetParam(Name="P1")
    param4 = model.GetParam(Name="P4")
    chart = model.GetChart(Name="Correlation Matrix 1")
    chart.EnableParameters(Parameters=[param1;param4], IsEnabled=false)
    chart.EnableParameters()

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName  The exported file name.

    Type       string

Optional Arguments

AppendMode  True to append to an existing csv file, False to overwrite it.

    Type       bool

    Default Value       False

Example

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

    container = system1.GetContainer(ComponentName="Design of Experiment")
    model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)

**Update**

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

**Example**

The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

**StatisticsChart**

The StatisticsChart shows the Probability Distribution Function and Cumulative Distribution Function results of a Six Sigma analysis

**Properties**

**DisplayParameterFullName**

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

*Type* bool

*Read Only* No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type* string

*Read Only* No

**FittingDistributionType**

Statistics chart plot type.

*Type* CdfPlotType

*Read Only* No

**IsUpToDate**

True if the entity is up-to-date.
Methods

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName The exported file name.

Type string

Optional Arguments

AppendMode True to append to an existing csv file, False to overwrite it.

Type bool

Default Value False

Example

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

Update

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

Example

The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

TradeoffChart

The data entity which describes the Tradeoff chart. It allows you to visualize the samples used in an optimization study and the Pareto fronts associated with them, if any. It supports the exploration of the generated samples in a 2D or 3D Mode, and can filter the number of visible Pareto Fronts.
Properties

Axes

Dictionary of the parameters associated to axes.

Type: Dictionary<ChartAxes, DataReference>

Read Only: Yes

DisplayParameterFullName

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

Type: bool

Read Only: No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string

Read Only: No

IsUpToDate

True if the entity is up-to-date.

Type: bool

Read Only: No

Mode

Chart mode, either 2D or 3D.

Type: TradeoffChartModes

Read Only: No

NumberOfParetoFront

Number of Pareto front to display. This is used as a filter to display only the most interesting fronts, given an optimization study.

Type: uint

Read Only: No

ShowInfeasiblePoints

If True, any infeasible points are displayed on the chart.
Methods

AssociateParameterToAxis

Associates a Parameter to the specified axis of the chart. The Parameter argument can be omitted which means that the axis is not set.

Required Arguments

Axis Axis to modify.

Type ChartAxes

Optional Arguments

Parameter Parameter entity to be assigned to the specified axis.

Type DataReference

Example

The following example shows how to assign parameters to the axes of a DesignPointsCurves chart.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
inputParam1 = model.GetParam(Name="P1")
outputParam1 = model.GetParam(Name="P5")
outputParam2 = model.GetParam(Name="P6")
chart = model.GetChart(Name="Design Point vs Parameter 1")
chart.AssociateParameterToAxis(Parameter=inputParam1, Axis="XAxis")
chart.AssociateParameterToAxis(Parameter=outputParam1, Axis="YAxis")
chart.AssociateParameterToAxis(Parameter=outputParam2, Axis="YRightAxis")
```

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName The exported file name.

Type string

Optional Arguments

AppendMode True to append to an existing csv file, False to overwrite it.

Type bool

Default Value False
Example
The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

**Update**

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

Example
The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

**DX GDO Design of Experiment**

This container holds Design of Experiment data for a Goal Driven Optimization.

**Methods**

**GetModel**

Get the DataReference of the Model. An exception is thrown if the entity is not found.

**Return**

The DataReference of the Model.

**Type**

`DataReference`

Example
The following example shows how the user can get a Model to change one of its properties.

```python
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
dOEModel1.DOEType = "eDOETYPE_OSPD"
```

**Data Entities**

**DiscreteLevel**

The data entity which describes a Discrete Level of an Input Parameter.
**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**Index**

Zero-based Index of the discrete level in the list of levels of the owning parameter.

<table>
<thead>
<tr>
<th>Type</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**Value**

Value of the DiscreteLevel.

<table>
<thead>
<tr>
<th>Type</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**Methods**

**SetValue**

Sets the value of a discrete level entity. A discrete level can have an integer value (e.g. a number if holes, a number of turns, etc) or a string value (e.g. a material name or a geometry file name).

**Required Arguments**

**Value**  
Value set to the discrete level entity.

<table>
<thead>
<tr>
<th>Type</th>
<th>Object</th>
</tr>
</thead>
</table>

**Example**

The following example shows how to retrieve a discrete level from an input parameter and then change its value.

```plaintext
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DiscreteInputParameter = model.GetParameter(Name="P2")
level1 = DiscreteInputParameter.GetDiscreteLevel(Name="Level 1")
level1.SetValue( Value="2500" )
```

**DistributionChart**

The entity which describes a parameter's Distribution chart. This chart is always associated with an uncertainty parameter and allows you to visualize the statistical distribution defined or calculated for this parameter.
Properties

DisplayParameterFullName

If True, the legend of the chart contains the full name of the parameters. Otherwise it contains the short name such as "P1".

Type bool
Read Only No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type string
Read Only No

IsUpToDate

True if the entity is up-to-date.

Type bool
Read Only No

Parameter

The parameter entity associated with the chart.

Type DataReference
Read Only No

Methods

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName The exported file name.

Type string

Optional Arguments

AppendMode True to append to an existing csv file, False to overwrite it.

Type bool
Default Value False
Example

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

Update

Updates the chart by generating all results or data required to plot it. If the chart is already up-to-date, nothing is done by default.

Example

The following example shows how to update a Tradeoff chart. The same code applies to all other types of charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart = model.GetChart(Name="TradeoffChart 1")
chart.Update()
```

DOEModel

Entity which performs and manages the DesignXplorer Design Of Experiments Component

Properties

**CCDTemplateType**

Template Type for CCD algorithm.

**Type**  
CCDTemplateType

**Read Only**  
No

**CCDType**

Design Type for CCD algorithm.

**Type**  
CentralCompositeDesignType

**Read Only**  
No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No
**ExportDesignPoints**

If True and PreserveDesignPoints is True as well, export project for each preserved Design Points.

*Type* `bool`

*Read Only* No

**MaxNumCycles**

Maximum Number Of Cycles for OSFD algorithm.

*Type* `int`

*Read Only* No

**Method**

Optimization Method

*Type* `DataReference`

*Read Only* No

**MethodName**

Type of the Design of Experiments

*Type* `string`

*Read Only* No

**NumberOfRetries**

Indicates the number of times DX will try to update the failed design points.

*Type* `int`

*Read Only* No

**NumSamp**

Number of Samples for User-Defined OSFD algorithm.

*Type* `int`

*Read Only* No

**OSFDType**

Design Type for OSFD algorithm.

*Type* `OptimalSpaceFillingType`

*Read Only* No
`PreserveDesignPoints`
If True, preserve the Design Points at the project level after the component Update.

**Type**     bool

**Read Only**  No

`RandomGeneratorSeed`
Seed value for LHS and OSFD algorithm.

**Type**     int

**Read Only**  No

`RetainDesignPoints`
If True and PreserveDesignPoints is True as well, retain data for each preserved Design Points.

**Type**     bool

**Read Only**  No

`RetryDelay`
Indicates how much time will elapse between tries. This option is only applicable when NumberOfRetries is greater than 0, otherwise it has no effect.

**Type**     Quantity

**Read Only**  No

`SampType`
Samples Type for LHS and OSFD algorithm.

**Type**     NumSampType

**Read Only**  No

**Methods**

`CreateChart`
Creates a Chart entity. The chart must be updated once after its creation by invoking its Update method. Then changes to its properties will update the chart automatically.

**Return**      The DataReference of the Chart.

**Type**     DataReference
**Required Arguments**

**ChartType**  
Type of chart to be created. The possible values depend on the type of Model. For instance, a ResponseSurface model accepts Spider, LocalSensitivity and Response chart while a CorrelationModel accepts CorrelationMatrix, DeterminationMatrix and CorrelationScatter charts.

  **Type**  ChartType

**Optional Arguments**

**DisplayText**  
Displayed name of the chart. If not specified, a default name is applied, depending on the value of ChartType.

  **Type**  string

**Example**

The following example shows how to create a chart.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
TradeoffChart = model.CreateChart(ChartType="eChartTradeoff")
TradeoffChart.Update()
```

**DeleteCharts**

Deletes a list of Chart entities.

**Required Arguments**

**Charts**  
List of Chart entities to delete.

  **Type**  List<DataReference>

**Example**

The following example shows how to delete existing charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart1 = model.GetChart(Name="TradeoffChart 1")
chart2 = model.GetChart(Name="SamplesChart 1")
model.DeleteCharts(Charts=[chart1, chart2])
```

**DuplicateChart**

Duplicates a Chart entity. The chart must be updated once after its creation by invoking its Update method. Then changes to its properties will update the chart automatically.

**Return**  The DataReference of the Chart.

  **Type**  DataReference

**Required Arguments**

**Chart**  The source chart to duplicate.
Type \textit{DataReference}

**Optional Arguments**

**DisplayText**

Displayed name of the chart. If not specified, a default name is applied, depending on the value of ChartType.

\textbf{Type} \textit{string}

**TargetModel**

The model on which the duplicated chart is created. If this parameter is not set, the model of the source chart is used.

\textbf{Type} \textit{DataReference}

**TargetResponsePoint**

Parent TargetResponsePoint of the chart. If this parameter is not set, the response point of the source chart is used if applicable.

\textbf{Type} \textit{DataReference}

**Example**

The following example shows how to duplicate a chart.

```python
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
designPointsCurvesChart1 = dOEModel1.GetChart(Name="DesignPointsCurves")
chart1 = dOEModel1.DuplicateChart(Chart=designPointsCurvesChart1)
chart1.Update()
```

**GetChart**

Query to return the chart reference for a given model and chart name.

\textbf{Return} \textit{The DataReference of the chart.}

\textbf{Type} \textit{DataReference}

**Required Arguments**

**Name** Name of the chart.

\textbf{Type} \textit{string}

**GetParameter**

Get the DataReference of a Parameter. An exception is thrown if the entity is not found.

\textbf{Return} \textit{The DataReference of the Parameter.}

\textbf{Type} \textit{DataReference}

**Required Arguments**

**Name** Name of the Parameter.

\textbf{Type} \textit{string}
**Example**

The following example shows how the user can get a parameter of a model to change one of its properties.

```python
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
inputParameter1 = dOEModel1.GetParameter(Name="P1")
inputParameter1.LowerBound = 1
```

**GetParameters**

Get the DataReferences of the InputParameter and OutputParameter of the model. If the optional argument InputParameters is not specified, the query returns all parameters. If it is specified and True, the query returns only input parameters. If it is False, the query returns only output parameters.

**Return**

The DataReferences of the Parameters

**Type**  
DataReferenceSet

**Optional Arguments**

- **InputParameters**
  
  If True, the query returns only input parameters. If False, the query returns only output parameters. If the argument is omitted, the query returns all parameters.

  **Type**  
  bool

**Example**

The following example shows how the user can get the parameters of a model.

```python
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
parameters = dOEModel1.GetParameters()
```

**GetParametricTable**

Get the DataReference of ParametricTable. An exception is thrown if the entity is not found. Names of the tables generated internally are: "DesignPoints", "CorrelationMatrix", "CorrelationScatter", "MinDesignPoints", "MaxDesignPoints", "ResponsePoints", "DeterminationMatrix".

**Return**

The DataReference of the ParametricTable

**Type**  
DataReference

**Required Arguments**

- **Name**  
  Name of the ParametricTable

  **Type**  
  string

**Example**

The following example shows how the user can get a ParametricTable to add a new row and set values.
**system1 = GetSystem(Name="RSO")**
`designOfExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")`
`dOEModel1 = designofExperiment1.GetModel()`  
`parametricTable = dOEModel1.GetParametricTable(Name="DesignPoints")`  
`parametricTable.AddRow()`  
`parametricTable.SetCellValue(RowIndex=9,ColumnIndex=0,Value="2.1")`

### ImportDesignPoints

Import existing Design Point entities in a custom Design of Experiments model. If a Design Point is solved, the output parameter values are also imported.

**Optional Arguments**

- **DesignPoints**  
  List of the existing Design Point entities to import. If not specified, all the Design Point entities found in the Parametric container are imported.
  
  **Type**  
  `List<DataReference>`

- **ExpandRanges**  
  If true, the command expands the range of variation of the input parameters based on the imported design points. This is done by modifying automatically the upper and lower bounds of the input parameters. If false, the design points which are not contained in the actual variation ranges are not imported.
  
  **Type**  
  `bool`

  **Default Value**  
  `False`

**Example**

The following example shows how the user can import all design points or a list of design points into a Design of Experiments model.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
model.DOEType = "eDOETYPE_USER"
model.ImportDesignPoints()
designPoint0 = Parameters.GetDesignPoint(Name="0")
designPoint1 = Parameters.GetDesignPoint(Name="1")
designPoint2 = Parameters.GetDesignPoint(Name="2")
model.ImportDesignPoints(DesignPoints=[designPoint0, designPoint1, designPoint2])
model.ImportDesignPoints(DesignPoints=[designPoint1])
```

### ImportDesignPointsFromFile

Import Design Point values in a custom Design of Experiments model from a csv file.

**Required Arguments**

- **FileName**  
  The imported file name.
  
  **Type**  
  `string`

**Optional Arguments**

- **ExpandRanges**  
  If true, the command expands the range of variation of the input parameters based on the imported design points. This is done by modifying automatically the upper and lower
bounds of the input parameters. If false, the design points which are not contained in the actual variation ranges are not imported. This option is not supported in the context of a DOE for a Six Sigma Analysis.

**Type**

`bool`

**Default Value**

`False`

**Example**

The following example shows how the user can import design points from a valid csv file.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
model.DOETYPE = "eDOETYPE_USER"
model.ImportDesignPointsFromFile(FilePath="designs.csv", ExpandRanges=True)
```

---

**PreviewDesignPoints**

Previews the Design Points of a model, without actually updating them, so that the user can adjust settings before launching a long update operation. This command applies to a Design of Experiments model, or the Refinement Points of a Kriging Response Surface, or a Parameters Correlation model. The command returns the table of the generated points.

**Return**

The DataReference of the ParametricTable containing the generated data.

**Type**

`DataReference`

**Example**

The following example shows how to preview a DOE model.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DOEMatrix = model.PreviewDesignPoints()
```

---

**InputParameter**

The data entity which describes an Input Parameter in DesignXplorer.

**Properties**

**Attribute1**

First editable attribute of the distribution for an uncertainty parameter. The nature of the attribute depends on the distribution type. For instance, the first attribute of a Normal distribution is the Mean value.

**Type**

`double`

**Read Only**

`No`
Attribute2

Second editable attribute of the distribution for an uncertainty parameter. The nature of the attribute depends on the distribution type. For instance, the second attribute of a Normal distribution is the Standard Deviation value. Some distribution type do not have a second attribute.

Type               double
Read Only           No

ConstantValue

Constant value of the Parameter when it is disabled.

Type               Object
Read Only           No

DiscreteLevels

List of the discrete levels.

Type               List<DataReference>
Read Only           Yes

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type               string
Read Only           No

DistributionLowerBound

Distribution lower bound of the variation range for an uncertainty Parameter.

Type               double
Read Only           No

DistributionType

Distribution type for an uncertainty parameter.

Type               DistType
Read Only           No

DistributionUpperBound

Distribution upper bound of the variation range for an uncertainty Parameter.

Type               double
**Read Only**  No

**Enabled**

True if the Parameter is enabled for the current study.

**Type**  bool

**Read Only**  No

**Kurtosis**

Kurtosis value of the distribution for an uncertainty parameter.

**Type**  double

**Read Only**  Yes

**LowerBound**

Lower bound of the variation range for a Continuous Parameter.

**Type**  double

**Read Only**  No

**Mean**

Mean value of the distribution for an uncertainty parameter.

**Type**  double

**Read Only**  Yes

**Nature**

Nature of the Parameter.

**Type**  ParameterNature

**Read Only**  No

**NumberOfLevels**

Number of levels if the parameter nature is Discrete, or the parameter nature is Continuous and the UseManufacturableValues property is set to True.

**Type**  int

**Read Only**  Yes

**Skewness**

Skewness value of the distribution for an uncertainty parameter.

**Type**  double
**Read Only**  Yes

**StandardDeviation**

Standard deviation value of the distribution for an uncertainty parameter.

**Type**  double

**Read Only**  Yes

**Type**

Type of the Parameter, either a DesignVariable in a GDO context, or an UncertaintyVariable in a SixSigma Analysis context.

**Type**  SimulationType

**Read Only**  Yes

**Units**

Units

**Type**  string

**Read Only**  Yes

**UpperBound**

Upper bound of the variation range for a Continuous Parameter.

**Type**  double

**Read Only**  No

**UseManufacturableValues**

True to restrict the variation of the parameter to defined Manufacturable Values.

**Type**  bool

**Read Only**  No

**Methods**

**AddDiscreteLevel**

Adds a Discrete Level entity on a discrete input parameter. A discrete level can have an integer value (e.g. a number of holes, a number of turns, etc) or a string value (e.g. a material name or a geometry file name). The command has optional arguments to specify the Name of the level and its Index in the list of levels of the parameter. By default, the new level is added to the end of the list. The various discrete levels of an input parameter represent independent configurations of the project, processed in the order of their creation.

**Return**  The created entity.
**DataReference**

**Required Arguments**

**Value**  
The value of the discrete level. Value can be an integer or a string.

  **Type**  
  *Object*

**Optional Arguments**

**DisplayText**  
DisplayText of the created entity. If not specified, a default name of the form "Level [#]" is used.

  **Type**  
  *string*

**Index**  
The position of the new level in the list of discrete levels of the parameter. Index is zero-based. If it is not specified, the new level is appended to the list.

  **Type**  
  *int*

**Default Value**  
-1

**Example**

The following example shows how to add new discrete levels on a discrete input parameter. The third level is inserted between the two others.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DiscreteInputParameter = model.GetParameter(Name="P2")
DiscreteInputParameter.AddDiscreteLevel(Value=3000, DisplayText="Three thousand turns")
DiscreteInputParameter.AddDiscreteLevel(Value=2000, DisplayText="Two thousand turns")
DiscreteInputParameter.AddDiscreteLevel(Value=5000, Index="1")
```

**AddLevels**

Adds a list of levels to a continuous input parameter. Each level is a quantity or a real number corresponding to a manufacturable value. The list of levels forms a restriction filter used when post-processing the input parameter. If levels are added outside of the variation range, the lower and upper bounds are adjusted accordingly.

**Required Arguments**

**Levels**  
List of added levels.

  **Type**  
  *List<string>*

**Optional Arguments**

**Overwrite**  
True in order to overwrite the existing levels, False by default.

  **Type**  
  *bool*

**Example**

The following example shows how to overwrite the manufacturable values of an input parameter and how to define an additional value later.
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
InputParameter = model.GetParameter(Name="P1")
InputParameter.UseManufacturableValues = True
InputParameter.AddLevels(Layers="0.3 [mm]", "0.5 [mm]", "1e-3 [m]", Overwrite=True)
InputParameter.AddLevels(Layers="7 [mm]")

CreateOptimizationCriterion

Creates an OptimizationCriterion entity associated to a parameter.

Return

The DataReference of the OptimizationCriterion.

Type

DataReference

Required Arguments

Parameter

The parameter on which the criterion is created.

Type

DataReference

Example

The following example shows how to create an OptimizationCriterion entity.

container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
parameter1 = model.GetParameter(Name="P1")
optimizationCriterion = parameter1.CreateOptimizationCriterion()

DeleteDiscreteLevels

Deletes a list of levels from a discrete input parameter.

Required Arguments

DiscreteLevels

List of the DiscreteLevel entities to delete.

Type

List<DataReference>

Example

The following example shows how to add and then delete one or more levels from a discrete input parameter.

container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DiscreteInputParameter = model.GetParameter(Name="P1")
level1 = DiscreteInputParameter.AddDiscreteLevel(Value=3000, DisplayText="Three thousand turns")
level2 = DiscreteInputParameter.AddDiscreteLevel(Value=2000, DisplayText="Two thousand turns")
level3 = DiscreteInputParameter.AddDiscreteLevel(Value=5000, Index="1")
DiscreteInputParameter.DeleteDiscreteLevels(DiscreteLevels=[level1, level2])

DeleteLevels

Deletes a list of levels from a continuous input parameter.
Required Arguments

Indices  
Indices of the items to remove from the levels list

  Type  List<int>

Example
The following example shows how to add and then delete one or more levels from a continuous input parameter for which the UseManufacturableValues property is set to True.

```csharp
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
InputParameter = model.GetParameter(Name="P1")
InputParameter.UseManufacturableValues = True
InputParameter.AddLevels(Levels=["0.3 [mmm]", "0.5 [mm]", "1e-3 [m]"], Overwrite=True)
InputParameter.DeleteLevels( Indices=[0, 1] )
```

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

FileName  
The exported file name.

  Type  string

Optional Arguments

AppendMode  
True to append to an existing csv file, False to overwrite it.

  Type  bool

  Default Value  False

Example
The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```csharp
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

GetDiscreteLevel

Get a discrete level by name from an input parameter. The parameter's full and ordered list of discrete levels is available as its "DiscreteLevels" property.

Return  
The DataReference of the discrete level entity.

  Type  DataReference
**Required Arguments**

**Name**   The name of the discrete level.

**Type**   string

**Example**

The following example shows how the user can retrieve a discrete level of a discrete input parameter by its name.

```plaintext
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
DiscreteInputParameter1 = dOEModel1.GetParameter(Name="P1")
level = DiscreteInputParameter1.GetDiscreteLevel(Name="Level 1")
```

**GetOptimizationCriterion**

Get the DataReference of the OptimizationCriterion associated with a Parameter. An exception is thrown if the entity is not found.

**Return**   The DataReference of the OptimizationCriterion.

**Type**   DataReference

**Required Arguments**

**Parameter**   Parent Parameter.

**Type**   DataReference

**Example**

The following example shows how the user can get a OptimizationCriterion to change one of its properties.

```plaintext
system1 = GetSystem(Name="RSO")
optimization1 = system1.GetContainer(ComponentName="Optimization")
optimizationModel1 = optimization1.GetModel()
parameter3 = optimizationModel1.GetParameter(Name="P3")
optimizationCriterion = parameter3.GetOptimizationCriterion()
optimizationCriterion.ObjectiveType = "eGT_MinimumPossible"
```

**GetParameterStatistics**

Get the DataReference of the ParameterStatistics entity associated with a Parameter.

**Return**   The DataReference of the ParameterStatistics entity.

**Type**   DataReference

**Example**

The following example shows how the user can get a ParameterStatistics entity and examine its Mean property.

```plaintext
system1 = GetSystem(Name="SSA")
sixSigmaAnalysis1 = system1.GetContainer(ComponentName="Six Sigma Analysis")
sixSigmaModel1 = sixSigmaAnalysis1.GetModel()
```
parameter4 = sixSigmaModel1.GetParameter(Name="P4")
parameterStatistics1 = parameter4.GetParameterStatistics()
mean = parameterStatistics1.Mean

**OutputParameter**

Output parameter entity for DesignXplorer.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type** string
- **Read Only** No

**FTestFiltering**

F-Test Filtering (beta)

- **Type** bool
- **Read Only** No

**InheritFromModelSettings**

Determines whether the Maximum Predicted Relative Error defined at the Model level is applicable to the output parameter.

- **Type** bool
- **Read Only** No

**LowerBound**

Minimum value extracted from existing design points and/or sample sets.

- **Type** double
- **Read Only** Yes

**MaxPredictedRelativeError**

Maximum Relative Error targeted for an output parameter when refining with the Kriging algorithm. This is the maximum predicted relative error that is acceptable for the selected output parameter.

- **Type** double
- **Read Only** No

**PredictedRelativeError**

Current value of the Predicted Relative Error when refining with the Kriging algorithm
**Type**
`double`

**Read Only**
Yes

**TransformationType**
Transformation Type

**Type**
`TransformationType`

**Read Only**
No

**Units**
Units

**Type**
`string`

**Read Only**
Yes

**UpperBound**
Maximum value extracted from existing design points and/or sample sets.

**Type**
`double`

**Read Only**
Yes

**Methods**

**CreateOptimizationCriterion**

Create an OptimizationCriterion entity associated to a parameter.

**Return**
The DataReference of the OptimizationCriterion.

**Type**
`DataReference`

**Required Arguments**

**Parameter**
The parameter on which the criterion is created.

**Type**
`DataReference`

**Example**
The following example shows how to create an OptimizationCriterion entity.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
parameter1 = model.GetParameter(Name="P1")
optimizationCriterion = parameter1.CreateOptimizationCriterion()
```

**GetOptimizationCriterion**

Get the DataReference of the OptimizationCriterion associated with a Parameter. An exception is thrown if the entity is not found.
Return The DataReference of the OptimizationCriterion.

Type DataReference

Required Arguments

Parameter Parent Parameter.

Type DataReference

Example
The following example shows how the user can get a OptimizationCriterion to change one of its properties.

```plaintext
system1 = GetSystem(Name="RSO")
optimization1 = system1.GetContainer(ComponentName="Optimization")
optimizationModel1 = optimization1.GetModel()
parameter3 = optimizationModel1.GetParameter(Name="P3")
optimizationCriterion = parameter3.GetOptimizationCriterion()
optimizationCriterion.ObjectiveType = "eGT_MinimumPossible"
```

GetParameterStatistics

Get the DataReference of the ParameterStatistics entity associated with a Parameter.

Return The DataReference of the ParameterStatistics entity.

Type DataReference

Example
The following example shows how the user can get a ParameterStatistics entity and examine its Mean property.

```plaintext
system1 = GetSystem(Name="SSA")
sixSigmaAnalysis1 = system1.GetContainer(ComponentName="Six Sigma Analysis")
sixSigmaModel1 = sixSigmaAnalysis1.GetModel()
parameter4 = sixSigmaModel1.GetParameter(Name="P4")
parameterStatistics1 = parameter4.GetParameterStatistics()
mean = parameterStatistics1.Mean
```

SamplingMethod

Entity which wraps and manages the external Sampling Method for the Design of Experiments component

Properties

No Properties.
DX GDO Response Surface

This container holds Response Surface data for a Goal Driven Optimization.

Methods

GetModel

Get the DataReference of the Model. An exception is thrown if the entity is not found.

Return

The DataReference of the Model.

Type

DataReference

Example

The following example shows how the user can get a Model to change one of its properties.

```python
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModell = designofExperiment1.GetModel()
dOEModell.DOETYPE = "eDOETYPE_OSF6"
```

Data Entities

GoodnessOfFit

Entity which manages the Goodness Of Fit Information of a Response Surface for an Output Parameter

Properties

DiscreteExpressions

A Dictionary holding the values of all discrete input. Note: Discrete parameter values are level values, not indices.

Type

Dictionary<DataReference, Object>

Read Only

No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type

string

Read Only

No

IsUpToDate

True if the entity is up-to-date.

Type

bool
Read Only  No

Methods

CreateAdvancedReport

For the standard response surface only (Full second order Polynomials), creates an Advanced Goodness of Fit report for any direct output parameter.

Return  A string which contains the generated Advanced Goodness of Fit report.

Type  string

Required Arguments

Parameter  Parent Parameter of the report.

Type  DataReference

Optional Arguments

PlainTextFormat  Plain text formatting instead of HTML (default).

Type  bool

Default Value  False

Example

The following example shows how to create an Advanced Goodness of Fit report.

```python
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
gof1 = model.GetGoodnessOfFit(Name="GoodnessOfFit")
outputParameter1 = model.GetParameter(Name="P4")
report1 = gof1.CreateAdvancedReport(Parameter=outputParameter1)
```

SetDiscreteParameter

Sets the Expression of a discrete input parameter in a GoodnessOfFit and updates the associated output parameter values. The chart entities depending on the GoodnessOfFit are updated as well.

Required Arguments

DiscreteParameter  DataReference of the Discrete Input parameter.

Type  DataReference

Expression  Assigned Expression (discrete value).

Type  string

Example

The following example shows how to set the expression for one discrete parameter in an existing GoodnessOfFit.
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
gof = model.GetGoodnessOfFit(Name="Goodness Of Fit")
inputParameter1 = model.GetParameter(Name="P1")
gof.SetDiscreteParameter(DiscreteParameter=inputParameter1, Expression="15")

**Update**

Updates the goodness of fit and the results which depend on it. If the goodness of fit is already up to date, nothing is done unless the Force flag is set to True.

**Optional Arguments**

**Force**  
Set to true if the update operation of the goodness of fit point is required even if it's already up to date.

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>False</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how to update an existing GoodnessOfFit.

```python
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
gof1 = model.GetGoodnessOfFit(Name="GOF 1")
gof1.Update()
```

**InputParameter**

The data entity which describes an Input Parameter in DesignXplorer.

**Properties**

**Attribute1**

First editable attribute of the distribution for an uncertainty parameter. The nature of the attribute depends on the distribution type. For instance, the first attribute of a Normal distribution is the Mean value.

<table>
<thead>
<tr>
<th>Type</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**Attribute2**

Second editable attribute of the distribution for an uncertainty parameter. The nature of the attribute depends on the distribution type. For instance, the second attribute of a Normal distribution is the Standard Deviation value. Some distribution type do not have a second attribute.

<table>
<thead>
<tr>
<th>Type</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>
**ConstantValue**

Constant value of the Parameter when it is disabled.

**Type**  Object

**Read Only**  No

**DiscreteLevels**

List of the discrete levels.

**Type**  List<DataReference>

**Read Only**  Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  string

**Read Only**  No

**DistributionLowerBound**

Distribution lower bound of the variation range for an uncertainty Parameter.

**Type**  double

**Read Only**  No

**DistributionType**

Distribution type for an uncertainty parameter.

**Type**  DistType

**Read Only**  No

**DistributionUpperBound**

Distribution upper bound of the variation range for an uncertainty Parameter.

**Type**  double

**Read Only**  No

**Enabled**

True if the Parameter is enabled for the current study.

**Type**  bool

**Read Only**  No
**Kurtosis**

Kurtosis value of the distribution for an uncertainty parameter.

*Type*  
double

*Read Only*  
Yes

**LowerBound**

Lower bound of the variation range for a Continuous Parameter.

*Type*  
double

*Read Only*  
No

**Mean**

Mean value of the distribution for an uncertainty parameter.

*Type*  
double

*Read Only*  
Yes

**Nature**

Nature of the Parameter.

*Type*  
ParameterNature

*Read Only*  
No

**NumberOfLevels**

Number of levels if the parameter nature is Discrete, or the parameter nature is Continuous and the UseManufacturableValues property is set to True.

*Type*  
int

*Read Only*  
Yes

**Skewness**

Skewness value of the distribution for an uncertainty parameter.

*Type*  
double

*Read Only*  
Yes

**StandardDeviation**

Standard deviation value of the distribution for an uncertainty parameter.

*Type*  
double

*Read Only*  
Yes
**Type**

Type of the Parameter, either a DesignVariable in a GDO context, or an UncertaintyVariable in a SixSigma Analysis context.

**Type**  
`SimulationType`

**Read Only**  
Yes

**Units**

Units

**Type**  
`string`

**Read Only**  
Yes

**UpperBound**

Upper bound of the variation range for a Continuous Parameter.

**Type**  
`double`

**Read Only**  
No

**UseManufacturableValues**

True to restrict the variation of the parameter to defined Manufacturable Values.

**Type**  
`bool`

**Read Only**  
No

**Methods**

**AddDiscreteLevel**

Adds a Discrete Level entity on a discrete input parameter. A discrete level can have an integer value (e.g. a number of holes, a number of turns, etc) or a string value (e.g. a material name or a geometry file name). The command has optional arguments to specify the Name of the level and its Index in the list of levels of the parameter. By default, the new level is added to the end of the list. The various discrete levels of an input parameter represent independent configurations of the project, processed in the order of their creation.

**Return**  
The created entity.

**Type**  
`DataReference`

**Required Arguments**

**Value**  
The value of the discrete level. Value can be an integer or a string.

**Type**  
`Object`
Optional Arguments

**DisplayText**
DisplayText of the created entity. If not specified, a default name of the form "Level [#]" is used.

*Type*  
*string*

**Index**
The position of the new level in the list of discrete levels of the parameter. Index is zero-based. If it is not specified, the new level is appended to the list.

*Type*  
*int*

*Default Value*  
*-1*

**Example**
The following example shows how to add new discrete levels on a discrete input parameter. The third level is inserted between the two others.

```
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()  
DiscreteInputParameter = model.GetParameter(Name="P2")
DiscreteInputParameter.AddDiscreteLevel(Value=3000, DisplayText="Three thousand turns")
DiscreteInputParameter.AddDiscreteLevel(Value=2000, DisplayText="Two thousand turns")
DiscreteInputParameter.AddDiscreteLevel(Value=5000, Index="1")
```

**AddLevels**

Adds a list of levels to a continuous input parameter. Each level is a quantity or a real number corresponding to a manufacturable value. The list of levels forms a restriction filter used when post-processing the input parameter. If levels are added outside of the variation range, the lower and upper bounds are adjusted accordingly.

**Required Arguments**

**Levels**
List of added levels.

*Type*  
*List*  
*string>*

**Optional Arguments**

**Overwrite**
True in order to overwrite the existing levels, False by default.

*Type*  
*bool*

**Example**
The following example shows how to overwrite the manufacturable values of an input parameter and how to define an additional value later.

```
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()  
InputParameter = model.GetParameter(Name="P1")
InputParameter.UseManufacturableValues = True
InputParameter.AddLevels(Levels=["0.3 [mm]", "0.5 [mm]", "1e-3 [m]"], Overwrite=True)
InputParameter.AddLevels(Levels="7 [mm]")
```
**CreateOptimizationCriterion**

Creates an OptimizationCriterion entity associated to a parameter.

**Return**

The DataReference of the OptimizationCriterion.

**Type** DataReference

**Required Arguments**

**Parameter**

The parameter on which the criterion is created.

**Type** DataReference

**Example**

The following example shows how to create an OptimizationCriterion entity.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
parameter1 = model.GetParameter(Name="P1")
optimizationCriterion = parameter1.CreateOptimizationCriterion()
```

**DeleteDiscreteLevels**

Deletes a list of levels from a discrete input parameter.

**Required Arguments**

**DiscreteLevels**

List of the DiscreteLevel entities to delete.

**Type** List<DataReference>

**Example**

The following example shows how to add and then delete one or more levels from a discrete input parameter.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DiscreteInputParameter = model.GetParameter(Name="P1")
level1 = DiscreteInputParameter.AddDiscreteLevel(Value=3000, DisplayText="Three thousand turns")
level2 = DiscreteInputParameter.AddDiscreteLevel(Value=2000, DisplayText="Two thousand turns")
level3 = DiscreteInputParameter.AddDiscreteLevel(Value=5000, Index="1")
DiscreteInputParameter.DeleteDiscreteLevels(DiscreteLevels=[level1, level2])
```

**DeleteLevels**

Deletes a list of levels from a continuous input parameter.

**Required Arguments**

**Indices**

Indices of the items to remove from the levels list

**Type** List<int>
Example

The following example shows how to add and then delete one or more levels from a continuous input parameter for which the UseManufacturableValues property is set to True.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
InputParameter = model.GetParameter(Name="P1")
InputParameter.UseManufacturableValues = True
InputParameter.AddLevels(Levels=['"0.3 [mmm]"', '"0.5 [mm]"', '"1e-3 [m]"'], Overwrite=True)
InputParameter.DeleteLevels(Indices=[0, 1])
```

ExportData

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

Required Arguments

**FileName**  
The exported file name.

  **Type** string

Optional Arguments

**AppendMode**  
True to append to an existing csv file, False to overwrite it.

  **Type** bool

  **Default Value** False

Example

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

GetDiscreteLevel

Get a discrete level by name from an input parameter. The parameter's full and ordered list of discrete levels is available as its "DiscreteLevels" property.

**Return**  
The DataReference of the discrete level entity.

  **Type** DataReference

Required Arguments

**Name**  
The name of the discrete level.

  **Type** string
Example
The following example shows how the user can retrieve a discrete level of a discrete input parameter by its name.

```
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
DiscreteInputParameter1 = dOEModel1.GetParameter(Name="P1")
level = DiscreteInputParameter1.GetDiscreteLevel(Name="Level 1")
```

**GetOptimizationCriterion**

Get the DataReference of the OptimizationCriterion associated with a Parameter. An exception is thrown if the entity is not found.

**Return**
The DataReference of the OptimizationCriterion.

**Type**  DataReference

**Required Arguments**

**Parameter**  Parent Parameter.

**Type**  DataReference

Example
The following example shows how the user can get a OptimizationCriterion to change one of its properties.

```
system1 = GetSystem(Name="RSO")
optimization1 = system1.GetContainer(ComponentName="Optimization")
optimizationModel1 = optimization1.GetModel()
parameter3 = optimizationModel1.GetParameter(Name="P3")
optimizationCriterion = parameter3.GetOptimizationCriterion()
optimizationCriterion.ObjectiveType = "eGT_MinimumPossible"
```

**GetParameterStatistics**

Get the DataReference of the ParameterStatistics entity associated with a Parameter.

**Return**
The DataReference of the ParameterStatistics entity.

**Type**  DataReference

Example
The following example shows how the user can get a ParameterStatistics entity and examine its Mean property.

```
system1 = GetSystem(Name="SSA")
sixSigmaAnalysis1 = system1.GetContainer(ComponentName="Six Sigma Analysis")
sixSigmaModel1 = sixSigmaAnalysis1.GetModel()
parameter4 = sixSigmaModel1.GetParameter(Name="P4")
parameterStatistics1 = parameter4.GetParameterStatistics()
mean = parameterStatistics1.Mean
```
**MinMaxSearch**

The data entity which described the MinMax Search option of a Response Surface Component.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type* `string`
*Read Only* No

**Enabled**

If True, performs a min-max search performed when the response surface is built.

*Type* `bool`
*Read Only* No

**IsUpToDate**

True if the entity is up-to-date.

*Type* `bool`
*Read Only* No

**NumberInitialPoints**

Number of initial samples for the min-max search algorithm.

*Type* `int`
*Read Only* No

**NumberStartPoints**

Number of start points for the min-max search algorithm.

*Type* `int`
*Read Only* No

**OutputParameter**

Output parameter entity for DesignXplorer.
Properties

DisplayText
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string
Read Only: No

FTestFiltering
F-Test Filtering (beta)

Type: bool
Read Only: No

InheritFromModelSettings
Determines whether the Maximum Predicted Relative Error defined at the Model level is applicable to the output parameter.

Type: bool
Read Only: No

LowerBound
Minimum value extracted from existing design points and/or sample sets.

Type: double
Read Only: Yes

MaxPredictedRelativeError
Maximum Relative Error targeted for an output parameter when refining with the Kriging algorithm. This is the maximum predicted relative error that is acceptable for the selected output parameter.

Type: double
Read Only: No

PredictedRelativeError
Current value of the Predicted Relative Error when refining with the Kriging algorithm

Type: double
Read Only: Yes

TransformationType
Transformation Type
**Type**  
*TransformationType*  

**Read Only**  
No  

**Units**  
Units  

**Type**  
*string*  

**Read Only**  
Yes  

**UpperBound**  
Maximum value extracted from existing design points and/or sample sets.  

**Type**  
*double*  

**Read Only**  
Yes  

**Methods**  

**CreateOptimizationCriterion**  
Creates an OptimizationCriterion entity associated to a parameter.  

**Return**  
The DataReference of the OptimizationCriterion.  

**Type**  
*DataReference*  

**Required Arguments**  

**Parameter**  
The parameter on which the criterion is created.  

**Type**  
*DataReference*  

**Example**  
The following example shows how to create an OptimizationCriterion entity.  

```python  
container = system1.GetContainer(ComponentName="Optimization")  
model = container.GetModel()  
parameter1 = model.GetParameter(Name="P1")  
optimizationCriterion = parameter1.CreateOptimizationCriterion()  
```

**GetOptimizationCriterion**  
Get the DataReference of the OptimizationCriterion associated with a Parameter. An exception is thrown if the entity is not found.  

**Return**  
The DataReference of the OptimizationCriterion.  

**Type**  
*DataReference*  

**Required Arguments**  

**Parameter**  
Parent Parameter.
**Type**  DataReference

**Example**

The following example shows how the user can get a OptimizationCriterion to change one of its properties.

```plaintext
system1 = GetSystem(Name="RSO")
optimization1 = system1.GetContainer(ComponentName="Optimization")
optimizationModel1 = optimization1.GetModel()
parameter3 = optimizationModel1.GetParameter(Name="P3")
optimizationCriterion = parameter3.GetOptimizationCriterion()
optimizationCriterion.ObjectiveType = "eGT_MinimumPossible"
```

**GetParameterStatistics**

Get the DataReference of the ParameterStatistics entity associated with a Parameter.

**Return**

The DataReference of the ParameterStatistics entity.

**Type**  DataReference

**Example**

The following example shows how the user can get a ParameterStatistics entity and examine its Mean property.

```plaintext
system1 = GetSystem(Name="SSA")
sixSigmaAnalysis1 = system1.GetContainer(ComponentName="Six Sigma Analysis")
sixSigmaModel1 = sixSigmaAnalysis1.GetModel()
parameter4 = sixSigmaModel1.GetParameter(Name="P4")
parameterStatistics1 = parameter4.GetParameterStatistics()
mean = parameterStatistics1.Mean
```

**ParametricTable**

ParametricTable entity used to encapsulate most of the evaluation results in a convenient 2D matrix format.

**Properties**

**DimCol**

Number of columns.

**Type**  int

**Read Only**  Yes

**DimRow**

Number of rows.

**Type**  int

**Read Only**  Yes
**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

**Methods**

**AddRow**

Adds a row to the bottom of a ParametricTable entity.

**Optional Arguments**

**RowValues**  
New values for the row.

**Type**  
List<string>

**Example**

The following example shows how to make a DOE editable, to retrieve the table of design points and add a new row to it.

```plaintext
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
model.DOETYPE = "eDOETYPE_USER"
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.AddRow()
```

**DeleteRows**

Delete rows from a ParametricTable entity.

**Required Arguments**

**Indices**  
Indices of the rows to delete.

**Type**  
List<int>

**Example**

The following example shows how to delete rows from the DOEmatrix in a custom DOE context.

```plaintext
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.DeleteRows(Indices=[0,7,8,9])
```

**ExportData**

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.
Required Arguments

**FileName**  The exported file name.

*Type*  *string*

Optional Arguments

**AppendMode**  True to append to an existing csv file, False to overwrite it.

*Type*  *bool*

*Default Value*  False

Example

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
center = system1.GetContainer(ComponentName="Design of Experiment")
model = center.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

---

**GetCellValue**

Get the Value of a ParametricTable's cell. An exception is thrown if the entity is not found.

*Return*  The value of the cell.

*Type*  *string*

Required Arguments

**ColumnIndex**  ColumnIndex (zero-based) of the cell.

*Type*  *int*

**RowIndex**  RowIndex (zero-based) of the cell.

*Type*  *int*

Example

The following example shows how the user can get the value of the ParametricTable's cell [0,3].

```python
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
doEModel1 = designofExperiment1.GetModel()
parametricTable = doEModel1.GetParametricTable(Name="DesignPoints")
cellValue = parametricTable.GetCellValue(RowIndex=0,ColumnIndex=3)
```

---

**GetRowUpdateOrder**

Get the Update Order value of a row in a ParametricTable. An exception is thrown if the entity is not found.
**Return**

The value of the update order.

**Type**  double

**Required Arguments**

**RowIndex**  RowIndex (zero-based) of the cell.

**Type**  int

**Example**

The following example shows how to get the Update Order value of row [3] in the ParametricTable.

```csharp
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
parametricTable = dOEModel1.GetParametricTable(Name="DesignPoints")
updateOrderValue = parametricTable.GetRowUpdateOrder(RowIndex=3)
```

**GetRowValues**

Get the Values of a ParametricTable's row. An exception is thrown if the entity is not found.

**Return**

List of the values of the specified row.

**Type**  List<string>

**Required Arguments**

**RowIndex**  RowIndex (zero-based) of the row to retrieve.

**Type**  int

**Example**

The following example shows how the user can get the values of the 4th ParametricTable's row.

```csharp
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
parametricTable = dOEModel1.GetParametricTable(Name="DesignPoints")
rowValues = parametricTable.GetRowValues(RowIndex=3)
```

**OptimizeUpdateOrder**

Optimizes the Update Order of Design Points to minimize the number of modifications between two consecutive Design Points. This command applies to the "DesignPoints" ParametricTable of a Design of Experiments model or of a Response Surface in a manual refinement context. It also applies to the "VerificationPoints" ParametricTable of a Response Surface model.

**Example**

The following example shows how to optimize the update order of the DesignPoints table in a Design of Experiments model.

```csharp
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
```
model.DOEType = "eDOETYPE_USER"
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.OptimizeUpdateOrder()

**SetCellValue**

Sets the value of a ParametricTable cell. If the table is read-only, the command has no effect.

**Required Arguments**

- **ColumnIndex**  
  Zero-based column index of the cell.  
  
  *Type*  
  int

- **RowIndex**  
  Zero-based row index of the cell.  
  
  *Type*  
  int

- **Value**  
  New value of the cell.  
  
  *Type*  
  string

**Example**

The following example shows how to set the value of the DesignPoints table in a custom DOE context.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
model.DOEType = "eDOETYPE_USER"
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.AddRow()
DOEMatrix.SetCellValue(RowIndex=0, ColumnIndex=0, Value="12.5 [mm]"
```

**SetOutputValuesEditable**

Sets the values of the output parameters as Editable (Editable=True) or Calculated (Editable=False) for the complete table, or a set of rows specified by the RowIndices argument. This command is applicable to the "DesignPoints" ParametricTable of a Design of Experiments model in a custom context (DOETYPE is "eDOETYPE_USER" or "eDOETYPE_CUSTOM_OSD") or of a Response Surface model in a manual refinement context. It also applies to the "VerificationPoints" ParametricTable of a Response Surface model.

**Required Arguments**

- **Editable**  
  True to define output values as Editable, or False to mark output values as calculated.  
  
  *Type*  
  bool

**Optional Arguments**

- **RowIndices**  
  Optional list of row zero-based indices. If this argument is not specified, the command applies to the complete ParametricTable.  
  
  *Type*  
  List<int>
Example
The following example shows how to switch a DOE model to the custom mode and then set the output values as editable for the first three design points of the DOE matrix.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
model.DOETYPE = "eDOETYPE_USER"
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.SetOutputValuesEditable(Editable=True, RowIndices=[0,1,2])
```

**SetRowUpdateOrder**
Sets the Update Order value for a row in a ParametricTable. If the table doesn't support Update Order functionality, the command has no effect.

**Required Arguments**

- **RowIndex**
  
  Zero-based row index of the cell.

  **Type** int

- **UpdateOrder**
  
  New value of the update order.

  **Type** double

**Example**
The following example shows how to set the Update Order value of the DesignPoints table in a Design of Experiments model.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.SetRowUpdateOrder(RowIndex=0, Value="2.0")
```

**SetRowValues**
Sets the values of a ParametricTable's row. If the table is read-only, the command has no effect.

**Required Arguments**

- **RowIndex**
  
  Zero-based row index of the cell.

  **Type** int

- **RowValues**
  
  New values for the row.

  **Type** List<string>

**Example**
The following example shows how to set the values of the DesignPoints table in a custom DOE context.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
model.DOETYPE = "eDOETYPE_USER"
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.AddRow()
DOEMatrix.SetRowValues(RowIndex=0, RowValues=["value1", "value2", "value3"])
```
DOEMatrix.SetRowValues(RowIndex=0, RowValues=[ "12.5 [mm]", "1" ])

**SetUpdateOrderByRow**

Sets a value for the UpdateOrder property of all Design Points using sorting settings. This command applies to the "DesignPoints" ParametricTable of a Design of Experiments model or of a Response Surface in a manual refinement context. It also applies to the "VerificationPoints" ParametricTable of a Response Surface model.

**Optional Arguments**

**SortBy** Definition of the sort.

**Type** List<string>

**Example**

The following example shows how to set the update order of the DesignPoints table in a Design of Experiments model. The Design Points are sorted first by their values for the parameter P3 (in ascending order), and then by their values for the parameter P1 (in descending order).

```csharp
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DOEMatrix = model.GetParametricTable(Name="DesignPoints")
DOEMatrix.SetUpdateOrderByRow(SortBy=["P3:+", "P1:-"])
```

**UpdateRows**

Updates the design points held in rows from a ParametricTable entity and the results which depend on it.

**Required Arguments**

**Indices** Indices of the rows to update.

**Type** List<int>

**Example**

The following example shows how to update the design points from the VerificationPoints Table.

```csharp
responseSurface1 = system1.GetContainer(ComponentName="Response Surface")
responseSurfaceModel1 = responseSurface1.GetModel()
parametricTable1 = responseSurfaceModel1.GetParametricTable(Name="VerificationPoints")
parametricTable1.UpdateRows(Indices=[0,1])
```

**ResponseSurfaceModel**

Entity which performs and manages the DesignXplorer Response Surface Component

**Properties**

**Converged**

Convergence state
Design Exploration

Type `bool`

**Read Only** Yes

**CrowdingDistSeparationPercentage**

Crowding Distance Separation Percentage when refining with the Kriging algorithm

Type `double`

**Read Only** No

**CurrentRelativeError**

Current value of the Relative Error

Type `double`

**Read Only** Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type `string`

**Read Only** No

**ExportDesignPoints**

If True and PreserveDesignPoints is True as well, export project for each preserved Design Points.

Type `bool`

**Read Only** No

**FittingType**

Response Surface Type

Type `FittingType`

**Read Only** No

**GenerateVerificationPoints**

If True, generate verification points.

Type `bool`

**Read Only** No

**KernelVariationType**

Kernel Variation Type for the Kriging algorithm
**Type**  
*KernelVariationType*

**Read Only**  
No

**MaximumDepth**

Maximum Depth limit when refining with the Sparse Grid response surface

**Type**  
*int*

**Read Only**  
No

**MaximumRelativeErrorSparseGrid**

Maximum Relative Error targeted when refining with the Sparse Grid response surface

**Type**  
*double*

**Read Only**  
No

**MaxNumberRefinementPointsKriging**

Maximum Number Of Refinement Points that can be generated for refinement with the Kriging algorithm.

**Type**  
*int*

**Read Only**  
No

**MaxNumberRefinementPointsSparseGrid**

Maximum Number Of Refinement Points that can be generated for refinement with the Sparse Grid response surface

**Type**  
*int*

**Read Only**  
No

**MaxPredictedRelativeError**

Maximum Relative Error targeted for all input parameters when refining with the Kriging algorithm. This is the maximum predicted relative error that is acceptable for all parameters.

**Type**  
*double*

**Read Only**  
No

**NumberOfCells**

Number of Cells for the Neural Network algorithm

**Type**  
*int*

**Read Only**  
No
**NumberOfRetries**

Indicates the number of times DX will try to update the failed design points.

**Type** int

**Read Only** No

**NumberRefinementPoints**

Number Of existing Refinement Points

**Type** int

**Read Only** Yes

**NumberVerificationPoints**

Number of verification points to generate.

**Type** int

**Read Only** No

**OutputVarCombinations**

Output Variable Combinations when refining with the Kriging algorithm. Controls how output variables are considered in terms of Predicated Relative Error and determines the number of refinement points generated per iteration.

**Type** AdaptKrigOutType

**Read Only** No

**PredictedRelativeError**

Current value of the Predicted Relative Error when refining with the Kriging algorithm

**Type** double

**Read Only** Yes

**PreserveDesignPoints**

If True, preserve the Design Points at the project level after the component Update.

**Type** bool

**Read Only** No

**RefinementType**

Refinement Type

**Type** ResponseSurfaceRefinementType
Read Only    No

**RetainDesignPoints**

If True and PreserveDesignPoints is True as well, retain data for each preserved Design Points.

Type        bool
Read Only    No

**RetryDelay**

Indicates how much time will elapse between tries. This option is only applicable when NumberOfRetries is greater than 0, otherwise it has no effect.

Type        Quantity
Read Only    No

**Methods**

**CreateChart**

Creates a Chart entity. The chart must be updated once after its creation by invoking its Update method. Then changes to its properties will update the chart automatically.

**Return**    The DataReference of the Chart.

Type        DataReference

**Required Arguments**

**ChartType**    Type of chart to be created. The possible values depend on the type of Model. For instance, a ResponseSurface model accepts Spider, LocalSensitivity and Response chart while a CorrelationModel accepts CorrelationMatrix, DeterminationMatrix and CorrelationScatter charts.

Type        ChartType

**Optional Arguments**

**DisplayText**    Displayed name of the chart. If not specified, a default name is applied, depending on the value of ChartType.

Type        string

**Example**

The following example shows how to create a chart.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
TradeoffChart = model.CreateChart(ChartType="eChartTradeoff")
TradeoffChart.Update()
```
**CreateGoodnessOfFit**

Creates a GoodnessOfFit. The ParameterValues dictionary can be used to specify a value for some or all of the discrete input parameters.

**Return**

The created entity.

**Type** DataReference

**Optional Arguments**

- **DisplayText**
  
  DisplayText of the created entity. If not specified, a default name of the form "Goodness Of Fit [#]" is used.

  **Type** string

- **ParameterValues**
  
  The values for each discrete input parameter. If not specified, each parameter is initialized to the current level.

  **Type** IDictionary<DataReference, string>

**Example**

The following example shows how to create a GoodnessOfFit from an existing ResponseSurface model. The code retrieves the parameters P1 and P2 and then creates the response point by assigning a value to each of these parameters.

```plaintext
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
inputParameter1 = model.GetParameter(Name="P1")
inputParameter2 = model.GetParameter(Name="P2")
goodnessOfFit = model.CreateGoodnessOfFit( DisplayText="GOF1",
ParameterValues={inputParameter1: "2", inputParameter2: "15"})
```

**CreateResponsePoint**

Creates a ResponsePoint. The ParameterValues dictionary can be used to specify a value or a quantity for some or all of the input parameters. The output parameter values cannot be specified. They are evaluated automatically from the ResponseSurface model once it is updated. Several types of charts can only be created as children of a ResponsePoint. These charts depend on the ResponsePoint and use the same parameter values.

**Return**

The created entity.

**Type** DataReference

**Optional Arguments**

- **DisplayText**
  
  DisplayText of the created entity. If not specified, a default name of the form "Response Point [#]" is used.

  **Type** string

- **ParameterValues**
  
  The values for each input parameter. If not specified, each parameter is initialized to the middle of its variation range.

  **Type** IDictionary<DataReference, string>
Example
The following example shows how to create a ResponsePoint from an existing ResponseSurface model. The code retrieves the parameters P1 and P2 and then creates the response point by assigning a value to each of these parameters.

```python
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
inputParameter1 = model.GetParameter(Name="P1")
inputParameter2 = model.GetParameter(Name="P2")
responsePoint = model.CreateResponsePoint( DisplayText="Improved Design",
ParameterValues={inputParameter1: "2.01", inputParameter2: "15.5"})
```

DeleteCharts

Deletes a list of Chart entities.

**Required Arguments**

**Charts**  
List of Chart entities to delete.

**Type**  
List<DataReference>

Example

The following example shows how to delete existing charts.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart1 = model.GetChart(Name="TradeoffChart 1")
chart2 = model.GetChart(Name="SamplesChart 1")
model.DeleteCharts(Charts=[chart1, chart2])
```

DeleteGoodnessOfFit

Deletes a list of GoodnessOfFit entities and all the depending Chart entities.

**Required Arguments**

**GoodnessOfFit**  
DataReferences of the entities to delete

**Type**  
List<DataReference>

Example

The following example shows how to delete existing GoodnessOfFit.

```python
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
gof1 = model.GetGoodnessOfFit(Name="GOF 1")
gof5 = model.GetGoodnessOfFit(Name="GOF 5")
gof6 = model.GetGoodnessOfFit(Name="GOF 6")
model.DeleteGoodnessOfFit(GoodnessOfFit=[gof1, gof5, gof6])
```

DeleteResponsePoints

Deletes a list of ResponsePoint entities and all the depending Chart entities.
Required Arguments

**ResponsePoints**  
DataReferences of the entities to delete

*Type*  
List<DataReference>

Example

The following example shows how to delete existing ResponsePoints.

```python
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
rp1 = model.GetResponsePoint(Name="Response Point 1")
rp5 = model.GetResponsePoint(Name="Response Point 5")
rp6 = model.GetResponsePoint(Name="Response Point 6")
model.DeleteResponsePoints(ResponsePoints=[rp1, rp5, rp6])
```

**DuplicateChart**

Duplicates a Chart entity. The chart must be updated once after its creation by invoking its Update method. Then changes to its properties will update the chart automatically.

*Return*  
The DataReference of the Chart.

*Type*  
DataReference

Required Arguments

**Chart**  
The source chart to duplicate.

*Type*  
DataReference

Optional Arguments

**DisplayText**  
Displayed name of the chart. If not specified, a default name is applied, depending on the value of ChartType.

*Type*  
string

**TargetModel**  
The model on which the duplicated chart is created. If this parameter is not set, the model of the source chart is used.

*Type*  
DataReference

**TargetResponsePoint**  
Parent TargetResponsePoint of the chart. If this parameter is not set, the response point of the source chart is used if applicable.

*Type*  
DataReference

Example

The following example shows how to duplicate a chart.

```python
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOModel1 = designofExperiment1.GetModel()
designPointsCurvesChart1 = dOModel1.GetChart(Name="DesignPointsCurves")
chart1 = dOModel1.DuplicateChart(Chart=designPointsCurvesChart1)
chart1.Update()
```
DuplicateGoodnessOfFit

Duplicates a GoodnessOfFit.

Return

The created entity.

Type  DataReference

Required Arguments

GoodnessOfFit  DataReference of the GoodnessOfFit.

Type  DataReference

Optional Arguments

DisplayText  DisplayText of the created entity. If not specified, a default name of the form "Goodness Of Fit [#]" is used.

Type  string

TargetModel  The ResponseSurface model on which the goodness of fit is created. If this parameter is not set, the model of the source goodness of fit is used.

Type  DataReference

Example

The following example shows how to duplicate a GoodnessOfFit from an existing ResponseSurface model. The code retrieves the parameters P1 and P2 and then creates the response point by assigning a value to each of these parameters.

```
container = system1.GetContainer(ComponentName="Response Surface")
responseSurfaceModel1 = container.GetModel()
gof = responseSurfaceModel1.GetGoodnessOfFit(Name="GoodnessOfFit")
goodnessOfFit2 = model.DuplicateGoodnessOfFit(GoodnessOfFit=gof, DisplayText="GOF2")
goodnessOfFit2.Update()
```

DuplicateResponsePoint

Duplicates a ResponsePoint entity. The response point must be updated once after its creation by invoking its Update method. Then changes to its properties will update the response point automatically.

Return

The created entity.

Type  DataReference

Required Arguments

ResponsePoint  The source response point to duplicate.

Type  DataReference

Optional Arguments

DisplayText  DisplayText of the created entity. If not specified, a default name of the form "Response Point [#]" is used.
**DuplicateCharts**

True if the charts of the response point have to be duplicated.

**Type**  
`bool`

**Default Value**  
False

**TargetModel**

The model on which the duplicated response point is created. If this parameter is not set, the model of the source response point is used.

**Type**  
`DataReference`

**Example**

The following example shows how to duplicate a response point and all its charts.

```python
responseSurface1 = system1.GetContainer(ComponentName="Response Surface")
responseSurfaceModel1 = responseSurface1.GetModel()
responsePoint1 = responseSurfaceModel1.GetResponsePoint(Name="ResponsePoint")
responsePoint2 = responseSurfaceModel1.DuplicateResponsePoint(ResponsePoint=responsePoint1, DuplicateCharts=True)
responsePoint2.Update()
```

**GetChart**

Query to return the chart reference for a given model and chart name.

**Return**  
The DataReference of the chart.

**Type**  
`DataReference`

**Required Arguments**

**Name**  
Name of the chart.

**Type**  
`string`

**GetGoodnessOfFit**

Get the DataReference of a GoodnessOfFit entity associated with a Model. An exception is thrown if the entity is not found.

**Return**  
The DataReference of the GoodnessOfFit.

**Type**  
`DataReference`

**Required Arguments**

**Name**  
Name of the GoodnessOfFit.

**Type**  
`string`

**Example**

The following example shows how the user can get a GoodnessOfFit.

```python
system1 = GetSystem(Name="RSO")
responseSurface1 = system1.GetContainer(ComponentName="Response Surface")
```
responseSurfaceModel1 = responseSurface1.GetModel()
gof = responseSurfaceModel1.GetGoodnessOfFit(Name="GoodnessOfFit")

GetGoodnessOfFits

Get the DataReferences of the GoodnessOfFit entities associated with a Model.

Return

The DataReferences of the GoodnessOfFits.

Type DataReferenceSet

Example

The following example shows how to retrieve the GoodnessOfFit entities of a response surface model.

system1 = GetSystem(Name="RSO")
responseSurface1 = system1.GetContainer(ComponentName="Response Surface")
responseSurfaceModel1 = responseSurface1.GetModel()
gof = responseSurfaceModel1.GetGoodnessOfFits()

GetMinMaxSearch

Get the DataReference of the MinMaxSearch entity associated with a Response Surface model. An exception is thrown if the entity is not found.

Return

The DataReference of the MinMaxSearch entity.

Type DataReference

Example

The following example shows how the user can get the MinMaxSearch of a ResponseSurfaceModel to change one of its properties.

responseSurface1 = system1.GetContainer(ComponentName="Response Surface")
responseSurfaceModel1 = responseSurface1.GetModel()
minMaxSearch1 = responseSurfaceModel1.GetMinMaxSearch()
minMaxSearch1.NumberInitialPoints = 200

GetParameter

Get the DataReference of a Parameter. An exception is thrown if the entity is not found.

Return

The DataReference of the Parameter.

Type DataReference

Required Arguments

Name Name of the Parameter.

Type string

Example

The following example shows how the user can get a parameter of a model to change one of its properties.
GetParameters

Get the DataReferences of the InputParameter and OutputParameter of the model. If the optional argument InputParameters is not specified, the query returns all parameters. If it is specified and True, the query returns only input parameters. If it is False, the query returns only output parameters.

Return

The DataReferences of the Parameters

Type

DataReferenceSet

Optional Arguments

InputParameters

If True, the query returns only input parameters. If False, the query returns only output parameters. If the argument is omitted, the query returns all parameters.

Type

bool

Example

The following example shows how the user can get the parameters of a model.

```python
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
inputParameter1 = dOEModel1.GetParameter(Name="P1")
inputParameter1.LowerBound = 1
```

GetParametricTable

Get the DataReference of ParametricTable. An exception is thrown if the entity is not found. Names of the tables generated internally are: "DesignPoints", "CorrelationMatrix", "CorrelationScatter", "MinDesignPoints", "MaxDesignPoints", "ResponsePoints", "DeterminationMatrix".

Return

The DataReference of the ParametricTable

Type

DataReference

Required Arguments

Name

Name of the ParametricTable

Type

string

Example

The following example shows how the user can get a ParametricTable to add a new row and set values.

```python
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
parametricTable = dOEModel1.GetParametricTable(Name="DesignPoints")
parametricTable.AddRow()
```
GetResponsePoint

Get the DataReference of a ResponsePoint. An exception is thrown if the entity is not found.

Return

The DataReference of the ResponsePoint.

Type DataReference

Required Arguments

Name Name

Type string

Example

The following example shows how to retrieve an existing ResponsePoint.

    container = system1.GetContainer(ComponentName="Response Surface")
    model = container.GetModel()
    rp = model.GetResponsePoint(Name="Response Point 1")

GetResponsePoints

Get the DataReferences of the ResponsePoints.

Return

The DataReferences of the ResponsePoints

Type DataReferenceSet

Example

The following example shows how to retrieve the response points of a response surface model.

    system1 = GetSystem(Name="RSO")
    container = system1.GetContainer(ComponentName="Response Surface")
    model = container.GetModel()
    responsePoints = model.GetResponsePoints()

ImportRefinementPointsFromFile

Import Refinement Point values in the manual refinement table of a Response Surface model from a csv file.

Required Arguments

FileName The imported file name.

Type string

Example

The following example shows how the user can import refinement points from a valid csv file.

    container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
model.ImportRefinementPointsFromFile(FilePath="designs.csv")

**ImportVerificationPointsFromFile**

Import Verification Point values in the verification table of a Response Surface model from a csv file.

**Required Arguments**

**FileName**  The imported file name.

**Type**  string

**Example**
The following example shows how the user can import verification points from a valid csv file.

```python
container = system1.GetContainer(ComponentName="Response Surface")
model = container.GetModel()
model.ImportVerificationPointsFromFile(FilePath="designs.csv")
```

**PreviewDesignPoints**

Previews the Design Points of a model, without actually updating them, so that the user can adjust settings before launching a long update operation. This command applies to a Design of Experiments model, or the Refinement Points of a Kriging Response Surface, or a Parameters Correlation model. The command returns the table of the generated points.

**Return**
The DataReference of the ParametricTable containing the generated data.

**Type**  DataReference

**Example**
The following example shows how to preview a DOE model.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DOEMatrix = model.PreviewDesignPoints()
```

---

**DX Parameters Correlation**

This container holds data for a Parameter Correlation.

**Methods**

**GetModel**

Get the DataReference of the Model. An exception is thrown if the entity is not found.

**Return**
The DataReference of the Model.

**Type**  DataReference
Example
The following example shows how the user can get a Model to change one of its properties.

```python
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
dOEModel1.DOEType = "eDOETYPE_OSFD"
```

**Data Entities**

**CorrelationModel**

Entity which performs and manages the DesignXplorer Parameters Correlation Component

**Properties**

**AutoStopType**
Auto Stop Type

*Type*  
CorrelationAutoStopType

*Read Only*  
No

**Converged**
Convergence state

*Type*  
bool

*Read Only*  
Yes

**ConvergenceCheckFrequency**
Convergence Check Frequency

*Type*  
int

*Read Only*  
No

**DisplayText**
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type*  
string

*Read Only*  
No

**ExportDesignPoints**
If True and PreserveDesignPoints is True as well, export project for each preserved Design Points.

*Type*  
bool
Read Only  No

**LinearCorrelationType**

Correlation Type

Type  LinearCorrelationType
Read Only  No

**MeanValueAccuracy**

Mean Value Accuracy

Type  double
Read Only  No

**NumberOfRetries**

Indicates the number of times DX will try to update the failed design points.

Type  int
Read Only  No

**NumberSamples**

Number of Samples

Type  int
Read Only  No

**PreserveDesignPoints**

If True, preserve the Design Points at the project level after the component Update.

Type  bool
Read Only  No

**RestartMode**

True to reuse the samples already generated

Type  bool
Read Only  No

**RetainDesignPoints**

If True and PreserveDesignPoints is True as well, retain data for each preserved Design Points.

Type  bool
**Read Only**  No

**RetryDelay**
Indicates how much time will elapse between tries. This option is only applicable when NumberOfRetries is greater than 0, otherwise it has no effect.

**Type**  *Quantity*

**SampleSetSize**
Size of Generated Sample Set

**Type**  *int*

**StandardDeviationAccuracy**
Standard Deviation Accuracy

**Type**  *double*

**Methods**

**CreateChart**
Creates a Chart entity. The chart must be updated once after its creation by invoking its Update method. Then changes to its properties will update the chart automatically.

**Return**  The DataReference of the Chart.

**Type**  *DataReference*

**Required Arguments**

**ChartType**  Type of chart to be created. The possible values depend on the type of Model. For instance, a ResponseSurface model accepts Spider, LocalSensitivity and Response chart while a CorrelationModel accepts CorrelationMatrix, DeterminationMatrix and CorrelationScatter charts.

**Type**  *ChartType*

**Optional Arguments**

**DisplayText**  Displayed name of the chart. If not specified, a default name is applied, depending on the value of ChartType.

**Type**  *string*

**Example**
The following example shows how to create a chart.
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
TradeoffChart = model.CreateChart(ChartType="eChartTradeoff")
TradeoffChart.Update()

**DeleteCharts**

Deletes a list of Chart entities.

**Required Arguments**

**Charts**  List of Chart entities to delete.

  **Type** List<DataReference>

**Example**

The following example shows how to delete existing charts.

```
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart1 = model.GetChart(Name="TradeoffChart 1")
chart2 = model.GetChart(Name="SamplesChart 1")
model.DeleteCharts(Charts=[chart1, chart2])
```

**DuplicateChart**

Duplicates a Chart entity. The chart must be updated once after its creation by invoking its Update method. Then changes to its properties will update the chart automatically.

**Return**

The DataReference of the Chart.

  **Type** DataReference

**Required Arguments**

**Chart**  The source chart to duplicate.

  **Type** DataReference

**Optional Arguments**

**DisplayText**  Displayed name of the chart. If not specified, a default name is applied, depending on the value of ChartType.

  **Type** string

**TargetModel**  The model on which the duplicated chart is created. If this parameter is not set, the model of the source chart is used.

  **Type** DataReference

**TargetResponsePoint**  Parent TargetResponsePoint of the chart. If this parameter is not set, the response point of the source chart is used if applicable.

  **Type** DataReference
**Example**

The following example shows how to duplicate a chart.

```plaintext
designOfExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designOfExperiment1.GetModel()
designPointsCurvesChart1 = dOEModel1.GetChart(Name="DesignPointsCurves")
chart1 = dOEModel1.DuplicateChart(Chart=designPointsCurvesChart1)
chart1.Update()
```

**GetChart**

Query to return the chart reference for a given model and chart name.

**Return**

The DataReference of the chart.

**Type**

DataReference

**Required Arguments**

**Name** Name of the chart.

**Type** string

**GetParameter**

Get the DataReference of a Parameter. An exception is thrown if the entity is not found.

**Return**

The DataReference of the Parameter.

**Type** DataReference

**Required Arguments**

**Name** Name of the Parameter.

**Type** string

**Example**

The following example shows how the user can get a parameter of a model to change one of its properties.

```plaintext
system1 = GetSystem(Name="RSO")
designOfExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designOfExperiment1.GetModel()
inputParameter1 = dOEModel1.GetParameter(Name="P1")
inputParameter1.LowerBound = 1
```

**GetParameters**

Get the DataReferences of the InputParameter and OutputParameter of the model. If the optional argument InputParameters is not specified, the query returns all parameters. If it is specified and True, the query returns only input parameters. If it is False, the query returns only output parameters.

**Return**

The DataReferences of the Parameters
**DataReferenceSet**

**Optional Arguments**

**InputParameters** If True, the query returns only input parameters. If False, the query returns only output parameters. If the argument is omitted, the query returns all parameters.

*Type* bool

**Example**

The following example shows how the user can get the parameters of a model.

```plaintext
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEMo dell1 = designofExperiment1.GetModel()
parameters = dOEMo dell1.GetParameters()
```

**GetParametricTable**

Get the DataReference of ParametricTable. An exception is thrown if the entity is not found. Names of the tables generated internally are: "DesignPoints", "CorrelationMatrix", "CorrelationScatter", "MinDesignPoints", "MaxDesignPoints", "ResponsePoints", "DeterminationMatrix".

**Return** The DataReference of the ParametricTable

*Type* DataReference

**Required Arguments**

**Name** Name of the ParametricTable

*Type* string

**Example**

The following example shows how the user can get a ParametricTable to add a new row and set values.

```plaintext
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEMo dell1 = designofExperiment1.GetModel()
parametricTable = dOEMo dell1.GetParametricTable(Name="DesignPoints")
parametricTable.AddRow()
parametricTable.SetCellValue(RowIndex=9,ColumnIndex=0,Value="2.1")
```

**PreviewDesignPoints**

Previews the Design Points of a model, without actually updating them, so that the user can adjust settings before launching a long update operation. This command applies to a Design of Experiments model, or the Refinement Points of a Kriging Response Surface, or a Parameters Correlation model. The command returns the table of the generated points.

**Return** The DataReference of the ParametricTable containing the generated data.

*Type* DataReference
Example
The following example shows how to preview a DOE model.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DOEMatrix = model.PreviewDesignPoints()
```

**InputParameter**

The data entity which describes an Input Parameter in DesignXplorer.

**Properties**

**Attribute1**

First editable attribute of the distribution for an uncertainty parameter. The nature of the attribute depends on the distribution type. For instance, the first attribute of a Normal distribution is the Mean value.

Type double
Read Only No

**Attribute2**

Second editable attribute of the distribution for an uncertainty parameter. The nature of the attribute depends on the distribution type. For instance, the second attribute of a Normal distribution is the Standard Deviation value. Some distribution type do not have a second attribute.

Type double
Read Only No

**ConstantValue**

Constant value of the Parameter when it is disabled.

Type Object
Read Only No

**DiscreteLevels**

List of the discrete levels.

Type List<DataReference>
Read Only Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type string
**Read Only**  No

**DistributionLowerBound**
Distribution lower bound of the variation range for an uncertainty Parameter.

**Type**  double

**Read Only**  No

**DistributionType**
Distribution type for an uncertainty parameter.

**Type**  DistType

**Read Only**  No

**DistributionUpperBound**
Distribution upper bound of the variation range for an uncertainty Parameter.

**Type**  double

**Read Only**  No

**Enabled**
True if the Parameter is enabled for the current study.

**Type**  bool

**Read Only**  No

**Kurtosis**
Kurtosis value of the distribution for an uncertainty parameter.

**Type**  double

**Read Only**  Yes

**LowerBound**
Lower bound of the variation range for a Continuous Parameter.

**Type**  double

**Read Only**  No

**Mean**
Mean value of the distribution for an uncertainty parameter.

**Type**  double
Read Only Yes

**Nature**

Nature of the Parameter.

**Type** ParameterNature

**Read Only** No

**NumberOfLevels**

Number of levels if the parameter nature is Discrete, or the parameter nature is Continuous and the UseManufacturableValues property is set to True.

**Type** int

**Read Only** Yes

**Skewness**

Skewness value of the distribution for an uncertainty parameter.

**Type** double

**Read Only** Yes

**StandardDeviation**

Standard deviation value of the distribution for an uncertainty parameter.

**Type** double

**Read Only** Yes

**Type**

Type of the Parameter, either a DesignVariable in a GDO context, or an UncertaintyVariable in a SixSigma Analysis context.

**Type** SimulationType

**Read Only** Yes

**Units**

Units

**Type** string

**Read Only** Yes

**UpperBound**

Upper bound of the variation range for a Continuous Parameter.
**Type**

double

**Read Only**

No

**UseManufacturableValues**

True to restrict the variation of the parameter to defined Manufacturable Values.

**Type**

bool

**Read Only**

No

**Methods**

**AddDiscreteLevel**

Adds a Discrete Level entity on a discrete input parameter. A discrete level can have an integer value (e.g. a number of holes, a number of turns, etc) or a string value (e.g. a material name or a geometry file name). The command has optional arguments to specify the Name of the level and its Index in the list of levels of the parameter. By default, the new level is added to the end of the list. The various discrete levels of an input parameter represent independent configurations of the project, processed in the order of their creation.

**Return**

The created entity.

**Type**

DataReference

**Required Arguments**

**Value**

The value of the discrete level. Value can be an integer or a string.

**Type**

Object

**Optional Arguments**

**DisplayText**

DisplayText of the created entity. If not specified, a default name of the form "Level [#]" is used.

**Type**

string

**Index**

The position of the new level in the list of discrete levels of the parameter. Index is zero-based. If it is not specified, the new level is appended to the list.

**Type**

int

**Default Value**

-1

**Example**

The following example shows how to add new discrete levels on a discrete input parameter. The third level is inserted between the two others.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DiscreteInputParameter = model.GetParameter(Name="P2")
DiscreteInputParameter.AddDiscreteLevel(Value=3000, DisplayText="Three thousand turns")
DiscreteInputParameter.AddDiscreteLevel(Value=2000, DisplayText="Two thousand turns")
```
DiscreteInputParameter.AddDiscreteLevel (Value=5000, Index="1")

**AddLevels**

Adds a list of levels to a continuous input parameter. Each level is a quantity or a real number corresponding to a manufacturable value. The list of levels forms a restriction filter used when post-processing the input parameter. If levels are added outside of the variation range, the lower and upper bounds are adjusted accordingly.

**Required Arguments**

**Levels**  List of added levels.

  **Type**  List<string>

**Optional Arguments**

**Overwrite**  True in order to overwrite the existing levels, False by default.

  **Type**  bool

**Example**

The following example shows how to overwrite the manufacturable values of an input parameter and how to define an additional value later.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
InputParameter = model.GetParameter(Name="P1")
InputParameter.UseManufacturableValues = True
InputParameter.AddLevels(Levels=["0.3 [mm]", "0.5 [mm]", "1e-3 [m]", Overwrite=True)
InputParameter.AddLevels(Levels="7 [mm]")
```

**CreateOptimizationCriterion**

Creates an OptimizationCriterion entity associated to a parameter.

**Return**  The DataReference of the OptimizationCriterion.

  **Type**  DataReference

**Required Arguments**

**Parameter**  The parameter on which the criterion is created.

  **Type**  DataReference

**Example**

The following example shows how to create an OptimizationCriterion entity.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
parameter1 = model.GetParameter(Name="P1")
optimizationCriterion = parameter1.CreateOptimizationCriterion()```
**DeleteDiscreteLevels**

Deletes a list of levels from a discrete input parameter.

**Required Arguments**

**DiscreteLevels** List of the DiscreteLevel entities to delete.

**Type** List<DataReference>

**Example**

The following example shows how to add and then delete one or more levels from a discrete input parameter.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DiscreteInputParameter = model.GetParameter(Name="P1")
level1 = DiscreteInputParameter.AddDiscreteLevel(Value=3000, DisplayText="Three thousand turns")
level2 = DiscreteInputParameter.AddDiscreteLevel(Value=2000, DisplayText="Two thousand turns")
level3 = DiscreteInputParameter.AddDiscreteLevel(Value=5000, Index="1")
DiscreteInputParameter.DeleteDiscreteLevels(DiscreteLevels=[level1, level2])
```

**DeleteLevels**

Deletes a list of levels from a continuous input parameter.

**Required Arguments**

**Indices** Indices of the items to remove from the levels list

**Type** List<int>

**Example**

The following example shows how to add and then delete one or more levels from a continuous input parameter for which the UseManufacturableValues property is set to True.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
InputParameter = model.GetParameter(Name="P1")
InputParameter.UseManufacturableValues = True
InputParameter.AddLevels(Levels=["0.3 [mmm]", "0.5 [mm]", "1e-3 [m]"], Overwrite=True)
InputParameter.DeleteLevels(Indices=[0, 1])
```

**ExportData**

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

**Required Arguments**

**FileName** The exported file name.

**Type** string
Optional Arguments

**AppendMode**  True to append to an existing csv file, False to overwrite it.

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>False</td>
</tr>
</tbody>
</table>

**Example**
The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

**GetDiscreteLevel**
Get a discrete level by name from an input parameter. The parameter's full and ordered list of discrete levels is available as its "DiscreteLevels" property.

**Return**
The DataReference of the discrete level entity.

| Type   | DataReference |

**Required Arguments**

| Name   | The name of the discrete level. |

| Type   | string |

**Example**
The following example shows how the user can retrieve a discrete level of a discrete input parameter by its name.

```python
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
DiscreteInputParameter1 = dOEModel1.GetParameter(Name="P1")
level = DiscreteInputParameter1.GetDiscreteLevel(Name="Level 1")
```

**GetOptimizationCriterion**
Get the DataReference of the OptimizationCriterion associated with a Parameter. An exception is thrown if the entity is not found.

**Return**
The DataReference of the OptimizationCriterion.

| Type   | DataReference |

**Required Arguments**

| Parameter   | Parent Parameter. |
**Type** DataReference

**Example**
The following example shows how the user can get a OptimizationCriterion to change one of its properties.

```plaintext
system1 = GetSystem(Name="RSO")
optimization1 = system1.GetContainer(ComponentName="Optimization")
optimizationModel1 = optimization1.GetModel()
parameter3 = optimizationModel1.GetParameter(Name="P3")
optimizationCriterion = parameter3.GetOptimizationCriterion()
optimizationCriterion.ObjectiveType = "eGT_MinimumPossible"
```

**GetParameterStatistics**
Get the DataReference of the ParameterStatistics entity associated with a Parameter.

**Return**
The DataReference of the ParameterStatistics entity.

**Type** DataReference

**Example**
The following example shows how the user can get a ParameterStatistics entity and examine its Mean property.

```plaintext
system1 = GetSystem(Name="SSA")
sixSigmaAnalysis1 = system1.GetContainer(ComponentName="Six Sigma Analysis")
sixSigmaModel1 = sixSigmaAnalysis1.GetModel()
parameter4 = sixSigmaModel1.GetParameter(Name="P4")
parameterStatistics1 = parameter4.GetParameterStatistics()
mean = parameterStatistics1.Mean
```

**OutputParameter**
Output parameter entity for DesignXplorer.

**Properties**

**DisplayText**
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string
**Read Only** No

**FTestFiltering**
F-Test Filtering (beta)

**Type** bool
**Read Only** No
**InheritFromModelSettings**

Determines whether the Maximum Predicted Relative Error defined at the Model level is applicable to the output parameter.

**Type**  
bool

**Read Only**  
No

**LowerBound**

Minimum value extracted from existing design points and/or sample sets.

**Type**  
double

**Read Only**  
Yes

**MaxPredictedRelativeError**

Maximum Relative Error targeted for an output parameter when refining with the Kriging algorithm. This is the maximum predicted relative error that is acceptable for the selected output parameter.

**Type**  
double

**Read Only**  
No

**PredictedRelativeError**

Current value of the Predicted Relative Error when refining with the Kriging algorithm.

**Type**  
double

**Read Only**  
Yes

**TransformationType**

Transformation Type

**Type**  
TransformationType

**Read Only**  
No

**Units**

Units

**Type**  
string

**Read Only**  
Yes

**UpperBound**

Maximum value extracted from existing design points and/or sample sets.

**Type**  
double
Read Only  Yes

Methods

CreateOptimizationCriterion

Creates an OptimizationCriterion entity associated to a parameter.

Return

The DataReference of the OptimizationCriterion.

Type  DataReference

Required Arguments

Parameter  The parameter on which the criterion is created.

Type  DataReference

Example

The following example shows how to create an OptimizationCriterion entity.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
parameter1 = model.GetParameter(Name="P1")
optimizationCriterion = parameter1.CreateOptimizationCriterion()
```

GetOptimizationCriterion

Get the DataReference of the OptimizationCriterion associated with a Parameter. An exception is thrown if the entity is not found.

Return

The DataReference of the OptimizationCriterion.

Type  DataReference

Required Arguments

Parameter  Parent Parameter.

Type  DataReference

Example

The following example shows how the user can get a OptimizationCriterion to change one of its properties.

```python
system1 = GetSystem(Name="RSO")
osptimization1 = system1.GetContainer(ComponentName="Optimization")
optimizationModel1 = optimization1.GetModel()
parameter3 = optimizationModel1.GetParameter(Name="P3")
optimizationCriterion = parameter3.GetOptimizationCriterion()
optimizationCriterion.ObjectiveType = "eGT_MinimumPossible"
```

GetParameterStatistics

Get the DataReference of the ParameterStatistics entity associated with a Parameter.
Return  The DataReference of the ParameterStatistics entity.

Type  DataReference

Example
The following example shows how the user can get a ParameterStatistics entity and examine its Mean property.

system1 = GetSystem(Name="SSA")
sixSigmaAnalysis1 = system1.GetContainer(ComponentName="Six Sigma Analysis")
sixSigmaModel1 = sixSigmaAnalysis1.GetModel()
parameter4 = sixSigmaModel1.GetParameter(Name="P4")
parameterStatistics1 = parameter4.GetParameterStatistics()
mean = parameterStatistics1.Mean

DX Six Sigma Analysis
This container holds data for a Six Sigma Analysis.

Methods

GetModel
Get the DataReference of the Model. An exception is thrown if the entity is not found.

Return  The DataReference of the Model.

Type  DataReference

Example
The following example shows how the user can get a Model to change one of its properties.

system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
dOEModel1.DOEType = "eDOETYPE_OSFD"

Data Entities

InputParameter
The data entity which describes an Input Parameter in DesignXplorer.

Properties

Attribute1
First editable attribute of the distribution for an uncertainty parameter. The nature of the attribute depends on the distribution type. For instance, the first attribute of a Normal distribution is the Mean value.

Type  double
**Read Only**  No

**Attribute2**

Second editable attribute of the distribution for an uncertainty parameter. The nature of the attribute depends on the distribution type. For instance, the second attribute of a Normal distribution is the Standard Deviation value. Some distribution type do not have a second attribute.

*Type*  double  
*Read Only*  No

**ConstantValue**

Constant value of the Parameter when it is disabled.

*Type*  Object  
*Read Only*  No

**DiscreteLevels**

List of the discrete levels.

*Type*  List<DataReference>  
*Read Only*  Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type*  string  
*Read Only*  No

**DistributionLowerBound**

Distribution lower bound of the variation range for an uncertainty Parameter.

*Type*  double  
*Read Only*  No

**DistributionType**

Distribution type for an uncertainty parameter.

*Type*  DistType  
*Read Only*  No

**DistributionUpperBound**

Distribution upper bound of the variation range for an uncertainty Parameter.
**Type**

`double`

**Read Only**

No

**Enabled**

True if the Parameter is enabled for the current study.

**Type**

`bool`

**Read Only**

No

**Kurtosis**

Kurtosis value of the distribution for an uncertainty parameter.

**Type**

`double`

**Read Only**

Yes

**LowerBound**

Lower bound of the variation range for a Continuous Parameter.

**Type**

`double`

**Read Only**

No

**Mean**

Mean value of the distribution for an uncertainty parameter.

**Type**

`double`

**Read Only**

Yes

**Nature**

Nature of the Parameter.

**Type**

`ParameterNature`

**Read Only**

No

**NumberOfLevels**

Number of levels if the parameter nature is Discrete, or the parameter nature is Continuous and the UseManufacturableValues property is set to True.

**Type**

`int`

**Read Only**

Yes

**Skewness**

Skewness value of the distribution for an uncertainty parameter.
**Type**  
`double`

**Read Only**  
Yes

**StandardDeviation**

Standard deviation value of the distribution for an uncertainty parameter.

**Type**  
`double`

**Read Only**  
Yes

**Type**

Type of the Parameter, either a DesignVariable in a GDO context, or an UncertaintyVariable in a SixSigma Analysis context.

**Type**  
`SimulationType`

**Read Only**  
Yes

**Units**

Units

**Type**  
`string`

**Read Only**  
Yes

**UpperBound**

Upper bound of the variation range for a Continuous Parameter.

**Type**  
`double`

**Read Only**  
No

**UseManufacturableValues**

True to restrict the variation of the parameter to defined Manufacturable Values.

**Type**  
`bool`

**Read Only**  
No

**Methods**

**AddDiscreteLevel**

Adds a Discrete Level entity on a discrete input parameter. A discrete level can have an integer value (e.g. a number of holes, a number of turns, etc) or a string value (e.g. a material name or a geometry file name). The command has optional arguments to specify the Name of the level and its Index in the list of levels of the parameter. By default, the new level is added to the end of the list. The various discrete levels of an input parameter represent independent configurations of the project, processed in the order of their creation.
Return  
The created entity.

Type  
DataReference

Required Arguments

Value  
The value of the discrete level. Value can be an integer or a string.

Type  
Object

Optional Arguments

DisplayText  
DisplayText of the created entity. If not specified, a default name of the form "Level [#]" is used.

Type  
string

Index  
The position of the new level in the list of discrete levels of the parameter. Index is zero-based. If it is not specified, the new level is appended to the list.

Type  
int

Default Value  
-1

Example

The following example shows how to add new discrete levels on a discrete input parameter. The third level is inserted between the two others.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DiscreteInputParameter = model.GetParameter(Name="P2")
DiscreteInputParameter.AddDiscreteLevel(Value=3000, DisplayText="Three thousand turns")
DiscreteInputParameter.AddDiscreteLevel(Value=2000, DisplayText="Two thousand turns")
DiscreteInputParameter.AddDiscreteLevel(Value=5000, Index="1")
```

AddLevels

Adds a list of levels to a continuous input parameter. Each level is a quantity or a real number corresponding to a manufacturable value. The list of levels forms a restriction filter used when post-processing the input parameter. If levels are added outside of the variation range, the lower and upper bounds are adjusted accordingly.

Required Arguments

Levels  
List of added levels.

Type  
List<string>

Optional Arguments

Overwrite  
True in order to overwrite the existing levels, False by default.

Type  
bool
Example
The following example shows how to overwrite the manufacturable values of an input parameter and how to define an additional value later.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
InputParameter = model.GetParameter(Name="P1")
InputParameter.UseManufacturableValues = True
InputParameter.AddLevels(Levels=["0.3 [mm]", "0.5 [mm]", "1e-3 [m]"], Overwrite=True)
InputParameter.AddLevels(Levels="7 [mm]")
```

CreateOptimizationCriterion

Creates an OptimizationCriterion entity associated to a parameter.

**Return**
The DataReference of the OptimizationCriterion.

**Type**
DataReference

**Required Arguments**

**Parameter**
The parameter on which the criterion is created.

**Type**
DataReference

Example
The following example shows how to create an OptimizationCriterion entity.

```python
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
parameter1 = model.GetParameter(Name="P1")
optimizationCriterion = parameter1.CreateOptimizationCriterion()
```

DeleteDiscreteLevels

Deletes a list of levels from a discrete input parameter.

**Required Arguments**

**DiscreteLevels**
List of the DiscreteLevel entities to delete.

**Type**
List<DataReference>

Example
The following example shows how to add and then delete one or more levels from a discrete input parameter.

```python
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
DiscreteInputParameter = model.GetParameter(Name="P1")
level1 = DiscreteInputParameter.AddDiscreteLevel(Value=3000, DisplayText="Three thousand turns")
level2 = DiscreteInputParameter.AddDiscreteLevel(Value=2000, DisplayText="Two thousand turns")
level3 = DiscreteInputParameter.AddDiscreteLevel(Value=5000, Index="1")
DiscreteInputParameter.DeleteDiscreteLevels(DiscreteLevels=[level1, level2])
```
**DeleteLevels**

Deletes a list of levels from a continuous input parameter.

**Required Arguments**

**Indices**
Indices of the items to remove from the levels list

**Type**
`List<int>`

**Example**

The following example shows how to add and then delete one or more levels from a continuous input parameter for which the `UseManufacturableValues` property is set to True.

```csharp
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
InputParameter = model.GetParameter(Name="P1")
InputParameter.UseManufacturableValues = True
InputParameter.AddLevels(Levels=["0.3 [mmm]", "0.5 [mm]", "1e-3 [m]"], Overwrite=True)
InputParameter.DeleteLevels( Indices=[0, 1] )
```

**ExportData**

Export the data of a DesignXplorer entity to a csv file. The entity can be one of the DesignXplorer chart entities, a ParametricTable or an InputParameter entity.

**Required Arguments**

**FileName**
The exported file name.

**Type**
`string`

**Optional Arguments**

**AppendMode**
True to append to an existing csv file, False to overwrite it.

**Type**
`bool`

**Default Value**
`False`

**Example**

The following example shows how the user can export the table of Design Points and then the Parameters Parallel chart of a Design of Experiments component.

```csharp
container = system1.GetContainer(ComponentName="Design of Experiment")
model = container.GetModel()
parametricTable = model.GetParametricTable(Name="DesignPoints")
parametricTable.ExportData(FileName="doe.csv")
chart = model.GetChart(Name="DesignPointsParallel")
chart.ExportData(FileName="doe.csv", AppendMode=True)
```

**GetDiscreteLevel**

Get a discrete level by name from an input parameter. The parameter's full and ordered list of discrete levels is available as its "DiscreteLevels" property.
**Return**

The DataReference of the discrete level entity.

**Type**  
DataReference

**Required Arguments**

**Name**  
The name of the discrete level.

**Type**  
string

**Example**

The following example shows how the user can retrieve a discrete level of a discrete input parameter by its name.

```plaintext
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
DiscreteInputParameter1 = dOEModel1.GetParameter(Name="P1")
level = DiscreteInputParameter1.GetDiscreteLevel(Name="Level 1")
```

**GetOptimizationCriterion**

Get the DataReference of the OptimizationCriterion associated with a Parameter. An exception is thrown if the entity is not found.

**Return**

The DataReference of the OptimizationCriterion.

**Type**  
DataReference

**Required Arguments**

**Parameter**  
Parent Parameter.

**Type**  
DataReference

**Example**

The following example shows how the user can get a OptimizationCriterion to change one of its properties.

```plaintext
system1 = GetSystem(Name="RSO")
optimization1 = system1.GetContainer(ComponentName="Optimization")
optimizationModel1 = optimization1.GetModel()
parameter3 = optimizationModel1.GetParameter(Name="P3")
optimizationCriterion = parameter3.GetOptimizationCriterion()
optimizationCriterion.ObjectiveType = "eGT_MinimumPossible"
```

**GetParameterStatistics**

Get the DataReference of the ParameterStatistics entity associated with a Parameter.

**Return**

The DataReference of the ParameterStatistics entity.

**Type**  
DataReference

**Example**

The following example shows how the user can get a ParameterStatistics entity and examine its Mean property.
system1 = GetSystem(Name="SSA")
sixSigmaAnalysis1 = system1.GetContainer(ComponentName="Six Sigma Analysis")
sixSigmaModel1 = sixSigmaAnalysis1.GetModel()
parameter4 = sixSigmaModel1.GetParameter(Name="P4")
parameterStatistics1 = parameter4.GetParameterStatistics()
mean = parameterStatistics1.Mean

**OutputParameter**

Output parameter entity for DesignXplorer.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**FTestFiltering**

F-Test Filtering (beta)

- **Type**: bool
- **Read Only**: No

**InheritFromModelSettings**

Determines whether the Maximum Predicted Relative Error defined at the Model level is applicable to the output parameter.

- **Type**: bool
- **Read Only**: No

**LowerBound**

Minimum value extracted from existing design points and/or sample sets.

- **Type**: double
- **Read Only**: Yes

**MaxPredictedRelativeError**

Maximum Relative Error targeted for an output parameter when refining with the Kriging algorithm. This is the maximum predicted relative error that is acceptable for the selected output parameter.

- **Type**: double
- **Read Only**: No
**PredictedRelativeError**

Current value of the Predicted Relative Error when refining with the Kriging algorithm

**Type**  
**double**

**Read Only**  
Yes

**TransformationType**

Transformation Type

**Type**  
**TransformationType**

**Read Only**  
No

**Units**

Units

**Type**  
**string**

**Read Only**  
Yes

**UpperBound**

Maximum value extracted from existing design points and/or sample sets.

**Type**  
**double**

**Read Only**  
Yes

**Methods**

**CreateOptimizationCriterion**

Creates an OptimizationCriterion entity associated to a parameter.

**Return**  
The DataReference of the OptimizationCriterion.

**Type**  
**DataReference**

**Required Arguments**

**Parameter**  
The parameter on which the criterion is created.

**Type**  
**DataReference**

**Example**

The following example shows how to create an OptimizationCriterion entity.

```plaintext
container = system1.GetChild(ObjectName="Optimization")
model = container.GetChild(ObjectName="Model")
parameter1 = model.GetChild(ObjectName="P1")
optimizationCriterion = parameter1.CreateOptimizationCriterion()
```
GetOptimizationCriterion

Get the DataReference of the OptimizationCriterion associated with a Parameter. An exception is thrown if the entity is not found.

Return

The DataReference of the OptimizationCriterion.

Type

DataReference

Required Arguments

Parameter

Parent Parameter.

Type

DataReference

Example

The following example shows how the user can get a OptimizationCriterion to change one of its properties.

```plaintext
system1 = GetSystem(Name="RSO")
optimization1 = system1.GetContainer(ComponentName="Optimization")
optimizationModel1 = optimization1.GetModel()
parameter3 = optimizationModel1.GetParameter(Name="P3")
optimizationCriterion = parameter3.GetOptimizationCriterion()
optimizationCriterion.ObjectiveType = "eGT_MinimumPossible"
```

GetParameterStatistics

Get the DataReference of the ParameterStatistics entity associated with a Parameter.

Return

The DataReference of the ParameterStatistics entity.

Type

DataReference

Example

The following example shows how the user can get a ParameterStatistics entity and examine its Mean property.

```plaintext
system1 = GetSystem(Name="SSA")
sixSigmaAnalysis1 = system1.GetContainer(ComponentName="Six Sigma Analysis")
sixSigmaModel1 = sixSigmaAnalysis1.GetModel()
parameter4 = sixSigmaModel1.GetParameter(Name="P4")
parameterStatistics1 = parameter4.GetParameterStatistics()
mean = parameterStatistics1.Mean
```

SixSigmaModel

Entity which performs and manages the Six Sigma Analysis Component

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.
<table>
<thead>
<tr>
<th><strong>Type</strong></th>
<th><strong>Read Only</strong></th>
<th><strong>ExportDesignPoints</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>No</td>
<td>If True and PreserveDesignPoints is True as well, export project for each preserved Design Points.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Type</strong></th>
<th><strong>Read Only</strong></th>
<th><strong>NumberOfRetries</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>No</td>
<td>Indicates the number of times DX will try to update the failed design points.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Type</strong></th>
<th><strong>Read Only</strong></th>
<th><strong>NumSamp</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>No</td>
<td>Number of Samples</td>
</tr>
</tbody>
</table>

| **Type**               | **Read Only** | **PreserveDesignPoints** |
|-----------------------|---------------|If True, preserve the Design Points at the project level after the component Update. |

| **Type**               | **Read Only** | **RetainDesignPoints** |
|-----------------------|---------------|If True and PreserveDesignPoints is True as well, retain data for each preserved Design Points. |

| **Type**               | **Read Only** | **RetryDelay** |
|-----------------------|---------------|Indicates how much time will elapse between tries. This option is only applicable when NumberOfRetries is greater than 0, otherwise it has no effect. |

| **Type**               | **Read Only** | **SampleGenType** |
|-----------------------|---------------|Sampling Type |
**Type**  
SampleGenType

**Read Only**  
No

**Methods**

**CreateChart**

Creates a Chart entity. The chart must be updated once after its creation by invoking its Update method. Then changes to its properties will update the chart automatically.

**Return**  
The DataReference of the Chart.

**Type**  
DataReference

**Required Arguments**

**ChartType**  
Type of chart to be created. The possible values depend on the type of Model. For instance, a ResponseSurface model accepts Spider, LocalSensitivity and Response chart while a CorrelationModel accepts CorrelationMatrix, DeterminationMatrix and CorrelationScatter charts.

**Type**  
ChartType

**Optional Arguments**

**DisplayText**  
Displayed name of the chart. If not specified, a default name is applied, depending on the value of ChartType.

**Type**  
string

**Example**

The following example shows how to create a chart.

```plaintext
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
TradeoffChart = model.CreateChart(ChartType="eChartTradeoff")
TradeoffChart.Update()
```

**DeleteCharts**

Deletes a list of Chart entities.

**Required Arguments**

**Charts**  
List of Chart entities to delete.

**Type**  
List<DataReference>

**Example**

The following example shows how to delete existing charts.

```plaintext
container = system1.GetContainer(ComponentName="Optimization")
model = container.GetModel()
chart1 = model.GetChart(Name="TradeoffChart 1")
chart2 = model.GetChart(Name="SamplesChart 1")
```
model.DeleteCharts(Charts=[chart1, chart2])

**DuplicateChart**

Duplicates a Chart entity. The chart must be updated once after its creation by invoking its Update method. Then changes to its properties will update the chart automatically.

**Return**

The DataReference of the Chart.

**Type**

DataReference

**Required Arguments**

**Chart**

The source chart to duplicate.

**Type**

DataReference

**Optional Arguments**

**DisplayText**

Displayed name of the chart. If not specified, a default name is applied, depending on the value of ChartType.

**Type**

string

**TargetModel**

The model on which the duplicated chart is created. If this parameter is not set, the model of the source chart is used.

**Type**

DataReference

**TargetResponsePoint**

Parent TargetResponsePoint of the chart. If this parameter is not set, the response point of the source chart is used if applicable.

**Type**

DataReference

**Example**

The following example shows how to duplicate a chart.

```python
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
designPointsCurvesChart1 = dOEModel1.GetChart(Name="DesignPointsCurves")
chart1 = dOEModel1.DuplicateChart(Chart=designPointsCurvesChart1)
chart1.Update()
```

**GetChart**

Query to return the chart reference for a given model and chart name.

**Return**

The DataReference of the chart.

**Type**

DataReference

**Required Arguments**

**Name**

Name of the chart.

**Type**

string
GetParameter

Get the DataReference of a Parameter. An exception is thrown if the entity is not found.

**Return**

The DataReference of the Parameter.

**Type**  
DataReference

**Required Arguments**

**Name**  
Name of the Parameter.

**Type**  
string

**Example**

The following example shows how the user can get a parameter of a model to change one of its properties.

```py
system1 = GetSystem(Name="RSO")
designOfExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designOfExperiment1.GetModel()
inputParameter1 = dOEModel1.GetParameter(Name="P1")
inputParameter1.LowerBound = 1
```

GetParameters

Get the DataReferences of the InputParameter and OutputParameter of the model. If the optional argument InputParameters is not specified, the query returns all parameters. If it is specified and True, the query returns only input parameters. If it is False, the query returns only output parameters.

**Return**

The DataReferences of the Parameters

**Type**  
DataReferenceSet

**Optional Arguments**

**InputParameters**  
If True, the query returns only input parameters. If False, the query returns only output parameters. If the argument is omitted, the query returns all parameters.

**Type**  
bool

**Example**

The following example shows how the user can get the parameters of a model.

```py
system1 = GetSystem(Name="RSO")
designOfExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designOfExperiment1.GetModel()
parameters = dOEModel1.GetParameters()
```

GetParametricTable

Get the DataReference of ParametricTable. An exception is thrown if the entity is not found. Names of the tables generated internally are: "DesignPoints", "CorrelationMatrix", "CorrelationScatter", "MinDesignPoints", "MaxDesignPoints", "ResponsePoints", "DeterminationMatrix".
**Return**  
The DataReference of the ParametricTable

**Type**  
DataReference

**Required Arguments**

**Name**  
Name of the ParametricTable

**Type**  
string

**Example**

The following example shows how the user can get a ParametricTable to add a new row and set values.

```plaintext
system1 = GetSystem(Name="RSO")
designofExperiment1 = system1.GetContainer(ComponentName="Design of Experiment")
dOEModel1 = designofExperiment1.GetModel()
parametricTable = dOEModel1.GetParametricTable(Name="DesignPoints")
parametricTable.AddRow()
parametricTable.SetCellValue(RowIndex=9, ColumnIndex=0, Value="2.1")
```
Engineering Data

The container used by an Engineering Data component to maintain project data.

Methods

AddToProjectDefaults

Adds an item, to the list of items, that will be added to new projects by default.

The item must be a favorite before it can be added to project defaults.

Required Arguments

Source  The source of material to be added to the project defaults.

Type  string

Optional Arguments

ItemType  The EngineeringDataType of the item. This can be Material, Load, or BeamSection.

Type  string

Default Value  Material

Example

The following example shows how to add a material named Concrete to the project defaults.

```
installDir = r"C:\Program Files\ANSYS Inc\v121"
EngData.AddToProjectDefaults(Name="Concrete",
Source=installDir+r"\Addins\EngineeringData\Samples\General_Materials.xml"
ItemType="Material")
```

CreateMaterial

Adds a new material to the container.

Return  The material data reference of the new material.

Type  DataReference

Required Arguments

Name  The name to be used for this material.
Exports engineering data to the specified file.

The following type of file format is supported for export:

MatML 3.1 schema for Material(s)

**Required Arguments**

- **FilePath**
  
The target path for the Engineering Data file ("*.xml").
  
  **Type** string

- **Format**
  
The file format in which engineering data will be exported.
  
  **Type** string

- **UnitSystem**
  
The unit system in which engineering data will be exported.
  
  **Type** string

**Optional Arguments**

- **ApplyScaleOffset**
  
The flag to specify if the scale factor and offset value will be applied during export.
  
  **Type** bool

  **Default Value** False

- **IgnoreSuppressed**
  
The flag to specify if suppressed engineering data will be ignored during export.
  
  **Type** bool

  **Default Value** False

- **OverwriteTarget**
  
The flag to specify if the target file will be overwritten.
  
  **Type** bool

  **Default Value** False

- **ReplaceMaterial**
  
The flag to specify if the earlier material data will be replaced in case of similar material names.
  
  **Type** bool

  **Default Value** False

**Example**

template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem(Position="Default")
container = system.GetContainer(ComponentName="Engineering Data")
Format="MatML31", UnitSystem="MKS_STANDARD", ReplaceMaterial=true)
GetMaterial

Returns a material of a given name from a container.

The name matching is case insensitive. If a material is not found an exception is thrown.

Return The material that matches the specified name.

Type DataReference

Required Arguments

Name The name of the material. This argument is case insensitive.

Type string

Example

The following example creates a new Engineering Data system and queries the default material, Structural Steel.

```
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem(Position="Default")
container = system.GetContainer(ComponentName="Engineering Data")
structuralSteel = container.GetMaterial(Name="Structural Steel")
```

GetMaterials

Returns the list of materials in a container. If no materials are in the container, the list is empty.

Return A DataReferenceSet of the materials in the container.

Type DataReferenceSet

Import

Imports engineering data into an existing source from a specified source.

The following types of files are supported for import:

- Engineering Data libraries exported from Workbench 9.0 to 11.0 SP1
- Material(s) file following the MatML 3.1 schema
- Material(s) file generated by AUTODYN

Required Arguments

Source The source which contains the materials.

Type string

ImportMaterial

Reads the data for a single material into the requested container.

Return A DataReference for the material that was read.
**DataReference**

**Required Arguments**

**Name**  
The name of the material to read from the source.

  **Type**  
  `string`

**Source**  
The source which contains the requested material.

  **Type**  
  `string`

**Example**
This code shows how to read the Copper Alloy material from the provided samples, into Engineering Data to use in an analysis.

```python
ingestDir = r"C:\Program Files\ANSYS Inc\v121"
mat1 = engineeringData1.ReadMaterial(
    Name="Copper Alloy",
    Source=ingestDir+r"\Addins\EngineeringData\Samples\General_Materials.xml")
```

**RemoveFromProjectDefaults**

Removes an item, from the list of items, that are added to new projects by default.

**Required Arguments**

**Source**  
The path to the file containing the item to remove.

  **Type**  
  `string`

**Optional Arguments**

**ItemType**  
The EngineeringDataType of the item. This can be Material, Load, or BeamSection.

  **Type**  
  `string`

  **Default Value**  
  Material

**Example**
The following example shows how to remove Concrete from project defaults.

```python
installDir = r"C:\Program Files\ANSYS Inc\v121"
EngData.RemoveFromProjectDefaults(Name="Concrete",
    Source=installDir+r"\Addins\EngineeringData\Samples\General_Materials.xml"
    ItemType="Material")
```

**Data Entities**

**CurveFit**

The entity to store curve fitting information.
Properties

CurveFitData

The coefficients of material model that approximates the experimental test data.

Type DataReference
Read Only No

DependentPropertyColl

The collection of experimental test data, e.g., Uniaxial Test data (Engineering Strain Vs Engineering Stress).

Type DataReferenceSet
Read Only No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type string
Read Only No

UnitSystemWhenSolved

The unit system of the coefficients of material model.

Type string
Read Only No

Methods

AddTestData

Adds test data from a given curve fitting.

Required Arguments

TestData The test data property to add.

Type DataReference

Example

The following example loads a material with experimental test data and a Neo-Hookean hyperelastic property.

```python
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem()
engineeringData = system.GetContainer(ComponentName="Engineering Data")
neopreneRubber = engineeringData.ReadMaterial{
```
CopyCoefficients

Copies the fitted coefficients from the curve fitting to the property data provided.

Required Arguments

**Destination**  The property data that will receive the coefficients.

**Type**  DataReference

Example

The following example loads a material with experimental test data and a Neo-Hookean hyperelastic property.

```python
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem()
engineeringData = system.GetContainer(ComponentName="Engineering Data")
neopreneRubber = engineeringData.ReadMaterial(
    Name="Neoprene Rubber",
    Source="Hyperelastic_Materials.xml")
neoHookeanProperty = neopreneRubber.GetProperty(Name="Neo-Hookean")
neoHookeanPropertyData = neoHookeanProperty.GetPropertyData(Name="Neo-Hookean")
curveFit = neoHookeanPropertyData.CreateCurveFitting(
    Type="Neo-Hookean",
    Definition="")
uniaxialProperty = neopreneRubber.GetProperty(Name="Uniaxial Test Data")
curveFit.AddTestData(TestData=uniaxialProperty)
biaxialProperty = neopreneRubber.GetProperty(Name="Biaxial Test Data")
curveFit.AddTestData(TestData=biaxialProperty)
shearProperty = neopreneRubber.GetProperty(Name="Shear Test Data")
curveFit.AddTestData(TestData=shearProperty)
volumetricProperty = neopreneRubber.GetProperty(Name="Volumetric Test Data")
curveFit.AddTestData(TestData=volumetricProperty)
curveFit.RemoveTestData(TestData=uniaxialProperty)
curveFit.Solve()
curveFit.CopyCoefficients(Destination=neoHookeanPropertyData)
curveFit.Delete()
```

Delete

Deletes a curve fitting.
Example
The following example loads a material with experimental test data and a Neo-Hookean hyperelastic property.

```python
template = GetTemplate(TemplateName="EngData")
system = template.createSystem()
engineeringData = system.GetContainer(ComponentName="Engineering Data")
neopreneRubber = engineeringData.ReadMaterial{
    Name="Neoprene Rubber",
    Source="Hyperelastic_Materials.xml")
neoHookeanProperty = neopreneRubber.GetProperty(Name="Neo-Hookean")
neoHookeanPropertyData = neoHookeanProperty.GetPropertyData(Name="Neo-Hookean")
curveFit = neoHookeanPropertyData.CreateCurveFitting(
    Type="Neo-Hookean",
    Definition="")
uniaxialProperty = neopreneRubber.GetProperty(Name="Uniaxial Test Data")
curveFit.AddTestData(TestData=uniaxialProperty)
biaxialProperty = neopreneRubber.GetProperty(Name="Biaxial Test Data")
curveFit.AddTestData(TestData=biaxialProperty)
shearProperty = neopreneRubber.GetProperty(Name="Shear Test Data")
curveFit.AddTestData(TestData=shearProperty)
volumetricProperty = neopreneRubber.GetProperty(Name="Volumetric Test Data")
curveFit.AddTestData(TestData=volumetricProperty)
curveFit.RemoveTestData(TestData=uniaxialProperty)
curveFit.AddTestData(TestData=uniaxialProperty)
curveFit.Solve()
curveFit.CopyCoefficients(Destination=neoHookeanPropertyData)
curveFit.Delete()
```

GetChartData

Returns a generated graph data for the specified source data.

Valid source data are:

Material Property
Material Property Data

Return The graph data for the specified source data.

Type DataReference

RemoveTestData

Removes test data from a given curve fitting.

Required Arguments

TestData The test data property to add.

Type DataReference

Example
The following example loads a material with experimental test data and a Neo-Hookean hyperelastic property.

```python
template = GetTemplate(TemplateName="EngData")
system = template.createSystem()
engineeringData = system.GetContainer(ComponentName="Engineering Data")
neopreneRubber = engineeringData.ReadMaterial{
```
Solve

The curve fitting module will solve for the coefficients which most nearly approximate the experimental test data.

CurveFitData

The entity which contains data relevant to the curve fitting solution. It is possible to call GetData on this entity after the curve fitting solution to determine coefficients. Also for nonlinear fits the seed values can be set via this entity.

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type \text{string}

Read Only No

Methods

DeleteData

Delete a row from a tabular data sheet.

Required Arguments

Index Index of the row to delete.

Type int
Optional Arguments

SheetName Name of the sheet to access.

Type string

SheetQualifiers SheetQualifiers is used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of the Qualifier and its Value.

Type Dictionary<string, string>

Example
The following example illustrates the deletion of a row from a tabular data sheet.

```
# Create a new Engineering Data System and access Structural Steel

template1 = GetTemplate(TemplateName="EngData")
system1 = template1.CreateSystem()
enGINEERING_DATA1 = system1.GetContainer(ComponentName="Engineering Data")
matl1 = ENGINEERING_DATA1.GetMaterial(Name="Structural Steel")

# Delete the first row in the Density property

matlProp1 = matl1.GetProperty(Name="Density")
matlProp1.DeleteTabularDataRow(Index = 0)

# Delete the first row in the Coefficient of Thermal Expansion property with optional SheetName and SheetQualifiers

matlProp2 = matl1.GetProperty(Name="Coefficient of Thermal Expansion")
matlProp2.DeleteTabularDataRow(
    SheetName="Coefficient of Thermal Expansion",
    SheetQualifiers={"Definition Method": "Secant", "Behavior": "Isotropic"},
    Index = 0)
```

GetData

Returns the tabular data associated with the data entity.

Return The returned data in scalar, list, or dictionary format.

Type Object

Optional Arguments

AsDictionary If set to true, the data will be returned as a dictionary where the keys are variable names and the values are the data for each variable. If set to false, the data will be returned in scalar or list format without the variable names.

Type bool

Default Value False

ColumnMajor If set to true, the data will be returned in column-major order. If set to false, the data will be returned in row-major order.

Type bool

Default Value True
**EndIndex**
The end index for requesting a subset of the data (zero-based).

**Type** int

**Default Value** -2147483647

**SheetName**
Specifies the sheet name when the data contains multiple sheets.

**Type** string

**SheetQualifiers**
Used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of qualifiers and values.

**Type** Dictionary<string, string>

**StartIndex**
The start index for requesting a subset of the data (zero-based).

**Type** int

**Default Value** 0

**Variables**
Names of the variables for which data is requested (string or list of strings).

**Type** Object

**Example**
In this example, all data is requested for the given tabular data entity.

```python
tabData1.GetData()
```

In this example, all data is requested in row-major order.

```python
tabData1.GetData(ColumnMajor=False)
```

In this example, all data is requested in dictionary format.

```python
tabData1.GetData(AsDictionary=True)
```

In this example, data for variables Density and Temperature is requested in dictionary format.

```python
tabData1.GetData(Variables=["Density", "Temperature"], AsDictionary=True)
```

**SetData**
Set tabular data associated with the data entity.

**Optional Arguments**

**Data**
Sets the data using a dictionary form. The keys are the variable names and the values are the data. The use of this argument is mutually exclusive with "Values" and "Variables".
**Type**  
Dictionary&lt;string, List&lt;Object&gt;&gt;  

**Index**  
Specifies the starting location used to set the data (zero-based). A value of -1 indicates that the data should be appended to the existing data.

**Type**  
int  

**Default Value**  
0  

**SheetName**  
Specifies the sheet name when the data contains multiple sheets.

**Type**  
string  

**SheetQualifiers**  
Used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of qualifiers and values.

**Type**  
Dictionary&lt;string, string&gt;  

**Values**  
List of data values set in conjunction with the "Variables" parameter. This parameter and the "Data" parameter are mutually exclusive.

**Type**  
List&lt;List&lt;Object&gt;&gt;  

**Variables**  
Names of the variables for which data is being set. This parameter and and the "Data" parameter are mutually exclusive.

**Type**  
List&lt;string&gt;  

**Example**

```
# Create a new Engineering Data System and access Structural Steel

template1 = GetTemplate(TemplateName="EngData")
system1 = template1.CreateSystem()
engineeringData1 = system1.GetContainer(ComponentName="Engineering Data")
mat11 = engineeringData1.GetMaterial(Name="Structural Steel")

# Change the value of a simple single-valued property
matlProp1 = mat11.GetProperty(Name="Density")
matlProp1.SetData(
    Variables="Density",
    Values="8500 [kg m^-3]"
)

# Set Temperature-dependent data for Elasticity based on lists of variables and values.
matlProp2 = mat11.GetProperty(Name="Elasticity")
temperature = ["400 [K]", "600 [K]", "800 [K]"
E = ["2e5 [MPa]", "1.9e5 [MPa]", "1.6e5 [MPa]"
matlProp2.SetData(
    Variables = ["Temperature","Young's Modulus"],
    Values = [temperature, E])

# Change the Temperature for the second table entry.
matlProp2.SetData(
    Index = 1,
    Variables = "Temperature",
    Values = "625 [K]"
)

# Set a list for Poisson's Ratio starting at the second table entry.
```
matlProp2.SetData(
    Index = 1,
    Variables = "Poisson's Ratio",
    Values = [0.3, 0.3])
#
# Set Temperature-dependent property data for the Coefficient of Thermal Expansion
# using a dictionary. The dictionary key is the Variable name,
# followed by the list of values for the variable.
#
matlProp3 = matl1.GetProperty(Name="Coefficient of Thermal Expansion")
newData = {
    "Temperature": ["200 [F]", "400 [F]", "600 [F]", "800 [F]", "1000 [F]"],
    "Coefficient of Thermal Expansion": ["6.3e-6 [F^-1]", "7.0e-6 [F^-1]",
                                         "7.46e-6 [F^-1]", "7.8e-6 [F^-1]",
                                         "8.04e-6 [F^-1]"],
}
matlProp3.SetData(
    SheetName="Coefficient of Thermal Expansion",
    SheetQualifiers={"Definition Method": "Secant", "Behavior": "Isotropic"},
    Data = newData)

**SetQualifier**

Changes the values of a specific qualifier in a data table.

**Required Arguments**

- **Qualifier**
  - The Qualifier to Set.
  - **Type**  string

- **Value**
  - The new value.
  - **Type**  string

**Optional Arguments**

- **SheetName**
  - The name of the tabular data sheet that contains the qualifier.
  - **Type**  string

- **SheetQualifiers**
  - SheetQualifiers can be used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of the Qualifier and its Value.
  - **Type**  Dictionary<string, string>

- **VariableName**
  - The name of the Variable that contains the qualifier to be changed.
  - **Type**  string

- **VariableQualifiers**
  - VariableQualifiers can be used to pass in the qualifiers to select between multiple variables with the same name. This is a dictionary of the Qualifier and its Value.
  - **Type**  Dictionary<string, string>

**Example**

The following example changes the 'Derive From' setting within an Isotropic Elasticity material property to be "Bulk Modulus and Poisson's Ratio".

matl = engineeringData1.GetMaterial(Name="Structural Steel")
matlProp1 = matl1.GetProperty(Name="Elasticity")
matlProp1.SetQualifier("
Material

The entity to store material information.

Properties

Description

The description of the material.

Type  string
Read Only  No

DisplayName

The display name of the material.

Type  string
Read Only  No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type  string
Read Only  No

Source

The path of the material's source file (*.engd" or ".xml").

Type  string
Read Only  Yes

Methods

AddToFavorites

Adds an item to the Favorites item list.

CreateProperty

Includes a physical quantity or the constitutive relation for the physical response of a material. A property can be visualized as one or more tables of data made up of one or more dependent and independent variables.
The material property is created based on the specified optional parameters "Definition" and "Behavior".

**Return**  Created material properly.

**Type**  DataReference

**Required Arguments**

**Name**  The new material property name.

**Type**  string

**Optional Arguments**

**Behavior**  The optional string to identify the way in which a new material property will behave. Behavior of some material properties can be specified in different ways, e.g., Elasticity can be specified as Isotropic, Orthotropic or Anisotropic.

**Type**  string

**Definition**  The optional string to identify the way in which new material property will be defined. Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

**Type**  string

**Qualifiers**  The optional dictionary of a qualifier name and it's corresponding value.

**Type**  Dictionary<string, string>

**Example**

The following example creates a new Engineering Data system and adds Coefficient of Thermal Expansion and Elasticity material properties to the default material Structural Steel.

Coefficient of Thermal Expansion is created using Secant definition and Orthotropic behavior. Elasticity is created using Orthotropic behavior.

```csharp
ENGDTemplate = GetTemplate(TemplateName="EngData")
ENGDSystem = ENGDTemplate.CreateSystem(Position="Default")

ENGDContainer = ENGDSystem.GetContainer(ComponentName="Engineering Data")
StructSteel = ENGDContainer.GetMaterial(Name="Structural Steel")

# Create material properties
CTEProperty = StructSteel.CreateProperty(
    Name="Coefficient of Thermal Expansion",
    Definition="Secant",
    Behavior="Orthotropic")
OrthoElasProperty = StructSteel.CreateProperty(
    Name="Elasticity",
    Behavior="Orthotropic")
```

**Delete**

Deletes the material.
**Duplicate**

Duplicates the data in this material and returns a new material. The name of the new material will be appended with a numerical value to make it unique.

**Return** The material data reference of the duplicated material.

**Type** DataReference

**Required Arguments**

**TargetContainer** The duplicated material will be added to this container.

**Type** DataContainerReference

**GetProperty**

Returns a material property of a given name from the specified material.

**Return** The material property that matches the specified name.

**Type** DataReference

**Required Arguments**

**Name** The name of the material property.

**Type** string

**Example**

The following example creates new Engineering Data system and queries Coefficient of Thermal Expansion property of the default material Structural Steel.

```python
ENGDTemplate = GetTemplate(TemplateName="EngData")
ENGDSystem = ENGDTemplate.CreateSystem(Position="Default")

ENGDContainer = ENGDSystem.GetContainer(ComponentName="Engineering Data")
StructSteel = ENGDContainer.GetMaterial(Name="Structural Steel")

# Get material property
CTEProperty = StructSteel.GetProperty(Name="Coefficient of Thermal Expansion")
```

**IsSuppressed**

Checks if an entity is suppressed.

**Valid entities are:**

- Material
- Material Property
- Material Property Data

**Return** Returns true if the entity is suppressed, false otherwise.

**Type** bool
**IsValid**

Validates a material data and provides a message in case of invalid data.

**Return**

The flag that indicates if the material is valid.

- **Type** bool

**Optional Arguments**

- **Message** The validation failure message.
  - **Type** Output<string>

**Refresh**

This will repopulate the contents of a material with data from a given source. The NameInSource property of a Material will be used to find a match in the source to pull data from. In the event that NameInSource is not set, the DisplayName property will be used instead.

Note: This operation will cause destruction of any data currently in the material.

```csharp
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem(Position="Default")
container = system.GetContainer(ComponentName="Engineering Data")
structSteel = container.GetMaterial(Name="Structural Steel")
density = structSteel.GetProperty(Name="Density")
density.Delete()
structSteel.UpdateMaterial(Source="General_Materials.xml")
restoredDensity = structSteel.GetProperty(Name="Density")
restoredDensity.SetData(SheetName="", Index=0, Variables=["Density"], Values=["8000 [kg m^-3]"])```

**Required Arguments**

- **Source** The path of the source file.
  - **Type** string

**RemoveFromFavorites**

Removes an item from the Favorite items list.

**Optional Arguments**

- **Format** The format of the file that contains the item.
  - **Type** string

- **Name** The name of the item to delete.
  - **Type** string

- **Source** The location of the file that contains the item on disk.
  - **Type** string

- **Type** The type of the object to delete. This is either a Material or a Mixture.
**Type** EngineeringDataType

**Example**
The following example gets the list of favorites from Engineering Data. It then selects and deletes the "Gray Cast Iron" material from the list.

```python
favorites = EngData.LoadFavoriteItems()
matl = favorites.GetMaterial(Name="Gray Cast Iron")
EngData.DeleteFromFavorites(
    Material=matl,
    Type="Material"
)
```

**SetAsDefaultFluidForModel**

This will specify the material to use (or not use) for parts in the model which are marked as a fluid.

If the material is in the Engineering Data component it will set or unset the material to be used in the model component of the system(s) that contain this Engineering Data component.

If the material is contained in Favorites it will set or unset the material to use as the default on fluids in the Engineering Data component when a new system is added to the project.

The material is set as the default in Favorites. It will automatically be added to the list of project defaults.

**Optional Arguments**

**Default**  
This Boolean is used to set or unset the material as the default.

**Type**  
bool

**Default Value**  
True

**SetAsDefaultSolidForModel**

This will specify the material to use (or not use) for parts in the model which are marked as a solid.

If the material is in the Engineering Data component it will set or unset the material to be used in the model component of the system(s) that contain this Engineering Data component.

If the material is contained in Favorites it will set or unset the material to use as the default on solids in the Engineering Data component when a new system is added to the project.

The material is set as the default in Favorites. It will automatically be added to the list of project defaults.

**Optional Arguments**

**Default**  
This Boolean is used to set or unset the material as the default.

**Type**  
bool

**Default Value**  
True

**SetSuppression**

Suppresses or Unsuppresses item.
Item can be suppressed to prevent it from being sent to a downstream cell in the system.

Following items can be suppressed:

Material

Material property

Material property data

**Required Arguments**

**Suppressed**  The flag to specify if the item should be suppressed or unsuppressed.

  Type  bool

**Unlink**

Unlinks a material from its underlying source. Once this occurs, the material can no longer be restored to its original state and the material will no longer have a source.

**MaterialProperty**

The entity to store material property information.

A material property is the identifier for the singular information (for example, Density) that together with other properties defines or models the behavior of the material. A property is always defined by at least one table (tabular data), which could be singular. Some properties can contain a collection of tabular data (for example, Isotropic Elasticity).

**Properties**

**Behavior**

The string that defines the way in which the material property behaves, e.g., Elasticity has Isotropic, Orthotropic or Anisotropic behavior.

  Type  string
  Read Only  No

**Definition**

The definition of the material property. Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

  Type  string
  Read Only  No

**Description**

The description of the material property.

  Type  string
**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string  
  - **Read Only**: No

**PropertyDataColl**

The collection of tabular data that defines the material property.

- **Type**: DataReferenceSet  
  - **Read Only**: Yes

**TypeName**

The name of the material property.

- **Type**: string  
  - **Read Only**: No

**Methods**

**AddTestData**

Includes experimental test data for response function calculations.

- **Required Arguments**
  - **TestData**: The test data property to add.  
    - **Type**: DataReference

**BeginBatchUpdate**

Marks the start of a series of data modifications to a table of data, to improve performance.

**CreatePropertyData**

Include an additional property data for a material property. Some properties may have more than one property data to describe the material property.

The preferred method of adding a material property data is to use CreateMaterialProperty.

- **Return**: The new material propery data that was created.  
  - **Type**: DataReference
**Required Arguments**

**Name**  The new material property data name.

**Type**  string

**Optional Arguments**

**Behavior**  A string to identify how the new material property data will behave.

The behavior of some material properties can be specified in different ways, e.g., Elasticity can be specified as Isotropic, Orthotropic or Anisotropic.

**Type**  string

**Definition**  A string to identify how the new material property data will be defined.

Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

**Type**  string

**Qualifiers**  The optional dictionary of a qualifier name and it's corresponding value.

**Type**  Dictionary<string, string>

**Example**

The following example creates the material property data Orthotropic Secant Coefficient of Thermal Expansion on the material property Coefficient of Thermal Expansion. This example assumes the material Structural Steel has been obtained from the General Materials library.

Get the material property we are going to create a new material property data on.

```csharp
thermExpansionMatProp = structuralSteel.GetMaterialProperty(Name="Coefficient of Thermal Expansion")
```

Create the new material property data.

```csharp
orthoSecantThermExpansionMatPropData = thermExpansionMatProp.CreatePropertyData(
    Name="Coefficient of Thermal Expansion",
    Definition="Secant",
    Behavior="Orthotropic")
```

**Delete**

Deletes the material property.

**Optional Arguments**

**Behavior**  The optional string to specify the material property behavior.

Behavior of some material properties can be specified in different ways e.g. Elasticity can be specified as Isotropic, Orthotropic or Anisotropic.
Type: **string**

**Definition**
The optional string to specify the material property definition.

Some material properties are defined in different ways e.g. Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

Type: **string**

**DeleteData**
Delete a row from a tabular data sheet.

**Required Arguments**

**Index**
Index of the row to delete.

Type: **int**

**Optional Arguments**

**SheetName**
Name of the sheet to access.

Type: **string**

**SheetQualifiers**
SheetQualifiers is used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of the Qualifier and its Value.

Type: **Dictionary<string, string>**

**Example**
The following example illustrates the deletion of a row from a tabular data sheet.

```plaintext
# Create a new Engineering Data System and access Structural Steel

template1 = GetTemplate(TemplateName="EngData")
system1 = template1.CreateSystem()
engineeringData1 = system1.GetContainer(ComponentName="Engineering Data")
matl1 = engineeringData1.GetMaterial(Name="Structural Steel")

# Delete the first row in the Density property

matlProp1 = matl1.GetProperty(Name="Density")
matlProp1.DeleteTabularDataRow(Index = 0)

# Delete the first row in the Coefficient of Thermal Expansion property with optional SheetName and SheetQualifiers

matlProp2 = matl1.GetProperty(Name="Coefficient of Thermal Expansion")
matlProp2.DeleteTabularDataRow(
    SheetName="Coefficient of Thermal Expansion",
    SheetQualifiers={"Definition Method": "Secant", "Behavior": "Isotropic"},
    Index = 0)
```

**EndBatchUpdate**
Marks the completion of a series of data modifications.
**GetChartData**

Returns a generated graph data for the specified source data.

Valid source data are:

- Material Property
- Material Property Data

**Return** The graph data for the specified source data.

**Type** DataReference

**GetData**

Returns the tabular data associated with the data entity.

**Return** The returned data in scalar, list, or dictionary format.

**Type** Object

**Optional Arguments**

**AsDictionary** If set to true, the data will be returned as a dictionary where the keys are variable names and the values are the data for each variable. If set to false, the data will be returned in scalar or list format without the variable names.

**Type** bool

**Default Value** False

**ColumnMajor** If set to true, the data will be returned in column-major order. If set to false, the data will be returned in row-major order.

**Type** bool

**Default Value** True

**EndIndex** The end index for requesting a subset of the data (zero-based).

**Type** int

**Default Value** -2147483647

**SheetName** Specifies the sheet name when the data contains multiple sheets.

**Type** string

**SheetQualifiers** Used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of qualifiers and values.

**Type** Dictionary<string, string>

**StartIndex** The start index for requesting a subset of the data (zero-based).

**Type** int
Default Value

**Variables**
Names of the variables for which data is requested (string or list of strings).

**Type**
Object

**Example**
In this example, all data is requested for the given tabular data entity.

```python
tabData1.GetData()
```

In this example, all data is requested in row-major order.

```python
tabData1.GetData(ColumnMajor=False)
```

In this example, all data is requested in dictionary format.

```python
tabData1.GetData(AsDictionary=True)
```

In this example, data for variables Density and Temperature is requested in dictionary format.

```python
tabData1.GetData(Variables=['Density', 'Temperature'], AsDictionary=True)
```

**GetPropertyData**

Returns a property data of the specified material property.

The property data returned is based on specified optional parameters "Definition" and "Behavior".

**Return**
The material property data that matches specified type name, definition and behavior.

**Type**
DataReference

**Required Arguments**

**Name**
The material property data type name.

**Type**
string

**Optional Arguments**

**Behavior**
The optional string to specify the material property data behavior.

Behavior of some material properties can be specified in different ways, e.g., Elasticity can be specified as Isotropic, Orthotropic or Anisotropic.

**Type**
string

**Definition**
The optional string to specify the material property data definition.
Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

**Type**  
string

**Qualifiers**  
The optional dictionary of a qualifier name and it's corresponding value.

**Type**  
Dictionary<string, string>

**Example**  
The following example creates new Engineering Data system and queries Coefficient of Thermal Expansion property data of the default material Structural Steel.

```csharp
ENGDTemplate = GetTemplate(TemplateName="EngData")
ENGDSystem = ENGDTemplate.CreateSystem(Position="Default")
ENGDContainer = ENGDSystem.GetContainer(ComponentName="Engineering Data")
StructSteel = ENGDContainer.GetMaterial(Name="Structural Steel")

CTEProperty = StructSteel.GetProperty(Name="Coefficient of Thermal Expansion")

RefTempPropData = CTEProperty.GetPropertyData(
    Name="Reference Temperature",
    Definition="Secant",
    Behavior="Isotropic")
```

**IsSuppressed**  
Checks if an entity is suppressed.

Valid entities are:

- Material
- Material Property
- Material Property Data

**Return**  
Returns true if the entity is suppressed, false otherwise.

**Type**  
bool

**IsValid**  
Validates a material property and provides a message in case of invalid data.

**Return**  
The flag that indicates if the material property is valid.

**Type**  
bool

**Required Arguments**

**Material**  
The parent material of the property.
Optional Arguments

Message
Type: Output<string>
The validation failure message.

RemoveTestData
Excludes experimental test data for response function calculations.

Required Arguments

TestData
Type: DataReference
The test data property to add.

SetData
Set tabular data associated with the data entity.

Optional Arguments

Data
Sets the data using a dictionary form. The keys are the variable names and the values are the data. The use of this argument is mutually exclusive with "Values" and "Variables".
Type: Dictionary<string, List<Object>>

Index
Specifies the starting location used to set the data (zero-based). A value of -1 indicates that the data should be appended to the existing data.
Type: int
Default Value: 0

SheetName
Specifies the sheet name when the data contains multiple sheets.
Type: string

SheetQualifiers
Used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of qualifiers and values.
Type: Dictionary<string, string>

Values
List of data values set in conjunction with the "Variables" parameter. This parameter and the "Data" parameter are mutually exclusive.
Type: List<List<Object>>

Variables
Names of the variables for which data is being set. This parameter and the "Data" parameter are mutually exclusive.
Type: List<string>
Example

# Create a new Engineering Data System and access Structural Steel

template1 = GetTemplate(TemplateName="EngData")
system1 = template1.CreateSystem()
engineeringData1 = system1.GetContainer(ComponentName="Engineering Data")
matl1 = engineeringData1.GetMaterial(Name="Structural Steel")
#
# Change the value of a simple single-valued property
#
matlProp1 = matl1.GetProperty(Name="Density")
matlProp1.SetData(
    Variables="Density",
    Values="8500 [kg m^-3]"
)
#
# Set Temperature-dependent data for Elasticity based
# on lists of variables and values.

E = ["2e5 [MPa]", "1.9e5 [MPa]", "1.6e5 [MPa]"
matlProp2.SetData(
    Variables = ["Temperature","Young's Modulus"],
    Values = [temperature, E])
#
# Change the Temperature for the second table entry.
#
matlProp2.SetData(
    Index = 1,
    Variables = "Temperature",
    Values = "625 [K]"
)
#
# Set a list for Poisson's Ratio starting at the second table entry.
#
matlProp2.SetData(
    Index = 1,
    Variables = "Poisson's Ratio",
    Values = [0.3, 0.3])
#
# Set Temperature-dependent property data for the Coefficient of Thermal Expansion
# using a dictionary. The dictionary key is the Variable name,
# followed by the list of values for the variable.
#
matlProp3 = matl1.GetProperty(Name="Coefficient of Thermal Expansion")
newData = {"Temperature": ["200 [F]","400 [F]","600 [F]","800 [F]","1000 [F]"],
            "Coefficient of Thermal Expansion": ["6.3e-6 [F^-1]","7.0e-6 [F^-1]",
                                                "7.46e-6 [F^-1]","7.8e-6 [F^-1]",
                                                "8.04e-6 [F^-1]"]}
matlProp3.SetData(
    SheetName="Coefficient of Thermal Expansion",
    SheetQualifiers={"Definition Method": "Secant", "Behavior": "Isotropic"},
    Data = newData)

**SetQualifier**

Changes the values of a specific qualifier in a data table.

**Required Arguments**

**Qualifier**  The Qualifier to Set.

**Type**  string

**Value**  The new value.
**Type**  string

**Optional Arguments**

**SheetName**  The name of the tabular data sheet that contains the qualifier.

**Type**  string

**SheetQualifiers**  SheetQualifiers can be used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of the Qualifier and its Value.

**Type**  Dictionary\(<\text{string, string}>\)

**VariableName**  The name of the Variable that contains the qualifier to be changed.

**Type**  string

**VariableQualifiers**  VariableQualifiers can be used to pass in the qualifiers to select between multiple variables with the same name. This is a dictionary of the Qualifier and its Value.

**Type**  Dictionary\(<\text{string, string}>\)

**Example**

The following example changes the 'Derive From' setting within an Isotropic Elasticity material property to be "Bulk Modulus and Poisson's Ratio".

```python
matl1 = engineeringData1.GetMaterial(Name="Structural Steel")
matlProp1 = matl1.GetProperty(Name="Elasticity")
matlProp1.SetQualifier(
  SheetName="Elasticity",
  Qualifier="Derive from",
  Value="Bulk Modulus and Poisson's Ratio")
```

**SetSuppression**

Suppresses or Unsuppresses item.

Item can be suppressed to prevent it from being sent to a downstream cell in the system.

Following items can be suppressed:

- Material
- Material property
- Material property data

**Required Arguments**

**Suppressed**  The flag to specify if the item should be suppressed or unsuppressed.

**Type**  bool
**MaterialPropertyData**

The entity to store material property (tabular) data information. The material property data is a collection of material variable data.

**Properties**

**Behavior**

The behavior of the material variable tabular data. Some material properties can have different behavior, e.g., Elasticity has Isotropic, Orthotropic or Anisotropic behavior.

- **Type**: string
- **Read Only**: No

**Definition**

The definition of the material variable tabular data. Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

- **Type**: string
- **Read Only**: No

**DependentColl**

The collection of dependent variables in the material variable tabular data, e.g., Density.

- **Type**: DataReferenceSet
- **Read Only**: Yes

**Description**

The description of the material variable tabular data.

- **Type**: string
- **Read Only**: No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**IndependentColl**

The collection of independent variables in the material variable tabular data, e.g., Temperature.

- **Type**: DataReferenceSet
**PrimaryIndependent**
The primary independent variable in the material variable tabular data, e.g., Temperature.

**Type**  
`DataReference`

**RowsCount**
The number of data values for a variable in the material variable tabular data.

**Type**  
`int`

**TypeName**
The name of the material variable tabular data.

**Type**  
`string`

**VariableColl**
The collection of variables in the material variable tabular data.

**Type**  
`DataReferenceSet`

**Methods**

**BeginBatchUpdate**
Marks the start of a series of data modifications to a table of data, to improve performance.

**CreateCurveFitting**
Creates a curve fitting for a given property data.

**Return**
The curve fitting.

**Type**  
`DataReference`

**Required Arguments**

**Definition**
The definition of curve fitting to create. This must be a definition that is supported by an engineering data curve fitting.

i.e. 1 Parameter, 2 Parameter, 1st Order, 2nd Order

**Type**  
`string`
**Type**

The type of curve fitting to create. This must be a type that is supported by an engineering data curve fitting.

i.e. Neo-Hookean, Mooney-Rivlin, Ogden, Yeoh, Polynomial

**Example**

The following example loads a material with experimental test data and a Neo-Hookean hyperelastic property.

```python
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem()
engineeringData = system.GetContainer(ComponentName="Engineering Data")
neopreneRubber = engineeringData.ReadMaterial(
    Name="Neoprene Rubber",
    Source="Hyperelastic_Materials.xml")
neoHookeanProperty = neopreneRubber.GetProperty(Name="Neo-Hookean")
neoHookeanPropertyData = neoHookeanProperty.GetPropertyData(Name="Neo-Hookean")
curveFit = neoHookeanPropertyData.CreateCurveFitting(
    Type="Neo-Hookean",
    Definition="")
uniaxialProperty = neopreneRubber.GetProperty(Name="Uniaxial Test Data")
curveFit.AddTestData(TestData=uniaxialProperty)
biaxialProperty = neopreneRubber.GetProperty(Name="Biaxial Test Data")
curveFit.AddTestData(TestData=biaxialProperty)
shearProperty = neopreneRubber.GetProperty(Name="Shear Test Data")
curveFit.AddTestData(TestData=shearProperty)
volumetricProperty = neopreneRubber.GetProperty(Name="Volumetric Test Data")
curveFit.AddTestData(TestData=volumetricProperty)
curveFit.RemoveTestData(TestData=uniaxialProperty)
curveFit.AddTestData(TestData=uniaxialProperty)
curveFit.Solve()
curveFit.CopyCoefficients(Destination=neoHookeanPropertyData)
curveFit.Delete()
```

**CreateVariable**

Include an additional variable in the property data for a material property.

**Return**

The new variable data that was created.

**Type**

**DataReference**

**Required Arguments**

**Name**

The new variable data name.

**Type**

**string**

**Optional Arguments**

**Qualifiers**

The optional dictionary of a qualifier name and it's corresponding value.

**Type**

**Dictionary<string, string>**

**Delete**

Deletes the material property data.
**EndBatchUpdate**

Marks the completion of a series of data modifications.

**GetChartData**

Returns a generated graph data for the specified source data.

Valid source data are:

- Material Property
- Material Property Data

**Return**  The graph data for the specified source data.

**Type**  DataReference

**GetCurveFitting**

Query to return the DataReference to the CurveFit used by some PropertyData.

**Return**  CurveFit DataReference

**Type**  DataReference

**GetVariable**

Returns a material variable of a given name from a material property data.

**Return**  The requested variable.

**Type**  DataReference

**Required Arguments**

**Name**  The name of the variable.

**Type**  string

**Example**

The following example creates new Engineering Data system and queries Coefficient of Thermal Expansion property data of the default material Structural Steel.

```plaintext
ENGDTemplate = GetTemplate(TemplateName="EngData")
ENGDSystem = ENGDTemplate.CreateSystem(Position="Default")
ENGDContainer = ENGDSystem.GetContainer(ComponentName="Engineering Data")
StructSteel = ENGDContainer.GetMaterial(Name="Structural Steel")

# Get material
CTEProperty = StructSteel.GetProperty(Name="Coefficient of Thermal Expansion")
# Get material property data
CTEPropData = CTEProperty.GetPropertyData(
    Name="Coefficient of Thermal Expansion",
    Definition="Secant",
    Behavior="Isotropic")
# Get material variable
CTEVariable = CTEPropData.GetVariable(
```
IsSuppressed

Checks if an entity is suppressed.

Valid entities are:

- Material
- Material Property
- Material Property Data

Return

Returns true if the entity is suppressed, false otherwise.

Type  bool

IsValid

Validates a material property data and provides a message in case of invalid data.

Return

The flag that indicates if the material property data is valid.

Type  bool

Required Arguments

Material  The parent material of the material property data.

Type  DataReference

Optional Arguments

Message  The validation failure message.

Type  Output<string>

SetSuppression

Suppresses or Unsuppresses item.

Item can be suppressed to prevent it from being sent to a downstream cell in the system.

Following items can be suppressed:

- Material
- Material property
- Material property data

Required Arguments

Suppressed  The flag to specify if the item should be suppressed or unsuppressed.

Type  bool
**MaterialVariable**

The entity to store material variable information.

**Properties**

**DatumColl**

The collection of the material variable data.

- **Type**: DataReferenceSet
- **Read Only**: No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**IsDependent**

The flag that indicates if the material variable is dependent, e.g., Density is temperature dependent.

- **Type**: bool
- **Read Only**: No

**LowerBoundUnit**

The unit of the lower bound data value in the material variable data.

- **Type**: string
- **Read Only**: No

**LowerBoundValue**

The lower bound data value in the material variable data.

- **Type**: double
- **Read Only**: No

**MaxValue**

The maximum value limit of the material variable data.

- **Type**: double
- **Read Only**: No
### MinValue
The minimum value limit of the material variable data.

**Type**  
`double`

**Read Only**  
No

### Offset
The offset value of the material variable data.

**Type**  
`Quantity`

**Read Only**  
No

### Scale
The scale factor of the material variable data.

**Type**  
`double`

**Read Only**  
No

### TypeName
The name of the material variable.

**Type**  
`string`

**Read Only**  
No

### UniqueData
The collection of unique data values in the material variable data.

**Type**  
`DataReferenceSet`

**Read Only**  
No

### UpperBoundUnit
The unit of the upper bound data value in the material variable data.

**Type**  
`string`

**Read Only**  
No

### UpperBoundValue
The upper bound data value in the material variable data.

**Type**  
`double`

**Read Only**  
No
Methods

Delete

Deletes an additional variable in the property data from a material property.

IsValid

Validates a material variable data and provides a message in case of invalid data.

Return

The flag that indicates if the material variable is valid.

Type    bool

Required Arguments

Material

The parent material of the variable.

Type    DataReference

MaterialPropertyData

The parent material property data of the variable.

Type    DataReference

Optional Arguments

Message

The validation failure message.

Type    Output<string>

Example

template = GetTemplate(TemplateName="EngData") system = template.CreateSystem(Position="Default") container = system.GetContainer(ComponentName="Engineering Data") structSteel = container.GetMaterial(Name="Structural Steel") CTEProp = structSteel.GetProperty(Name="Coefficient of Thermal Expansion") CTEPropData = CTEProperty.GetPropertyData( Name="Coefficient of Thermal Expansion", Definition="Secant", Behavior="Isotropic") CTEVariable = CTEPropData.GetVariable( Name="Coefficient of Thermal Expansion") valid = EngData.ValidateMaterialVariable(MaterialVariable=CTEVariable, Material=structSteel, MaterialPropertyData=CTEPropdata)

--

Engineering Data Curve Fit

The container used by Engineering Data to maintain data for curve fitting operations.

Data Entities

CurveFit

The entity to store curve fitting information.

Properties

CurveFitData

The coefficients of material model that approximates the experimental test data.
The collection of experimental test data, e.g., Uniaxial Test data (Engineering Strain Vs Engineering Stress).

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

The unit system of the coefficients of material model.

The following example loads a material with experimental test data and a Neo-Hookean hyperelastic property.

```python
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem()
engineeringData = system.GetContainer(ComponentName="Engineering Data")
neopreneRubber = engineeringData.ReadMaterial(
    Name="Neoprene Rubber",
    Source="Hyperelastic_Materials.xml")
neoHookeanProperty = neopreneRubber.GetProperty(Name="Neo-Hookean")
neoHookeanPropertyData = neoHookeanProperty.GetPropertyData(Name="Neo-Hookean")
curveFit = neoHookeanPropertyData.CreateCurveFitting(
    Type="Neo-Hookean",
    Definition="")
uniaxialProperty = neopreneRubber.GetProperty(Name="Uniaxial Test Data")
```
curveFit.AddTestData(TestData=uniaxialProperty)
biaxialProperty = neopreneRubber.GetProperty(Name="Biaxial Test Data")
curveFit.AddTestData(TestData=biaxialProperty)
shearProperty = neopreneRubber.GetProperty(Name="Shear Test Data")
curveFit.AddTestData(TestData=shearProperty)
volumetricProperty = neopreneRubber.GetProperty(Name="Volumetric Test Data")
curveFit.AddTestData(TestData=volumetricProperty)
curveFit.RemoveTestData(TestData=uniaxialProperty)
curveFit.AddTestData(TestData=uniaxialProperty)
curveFit.Solve()
curveFit.CopyCoefficients(Destination=neoHookeanPropertyData)
curveFit.Delete()

**CopyCoefficients**

Copies the fitted coefficients from the curve fitting to the property data provided.

**Required Arguments**

**Destination**

The property data that will receive the coefficients.

**Type**

**DataReference**

**Example**

The following example loads a material with experimental test data and a Neo-Hookean hyperelastic property.

```python
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem()
engineeringData = system.GetContainer(ComponentName="Engineering Data")
neopreneRubber = engineeringData.ReadMaterial(
    Name="Neoprene Rubber",
    Source="Hyperelastic_Materials.xml")
neoHookeanProperty = neopreneRubber.GetProperty(Name="Neo-Hookean")
neoHookeanPropertyData = neoHookeanProperty.GetPropertyData(Name="Neo-Hookean")
curveFit = neoHookeanPropertyData.CreateCurveFitting(
    Type="Neo-Hookean",
    Definition="")
uniaxialProperty = neopreneRubber.GetProperty(Name="Uniaxial Test Data")
curveFit.AddTestData(TestData=uniaxialProperty)
biaxialProperty = neopreneRubber.GetProperty(Name="Biaxial Test Data")
curveFit.AddTestData(TestData=biaxialProperty)
shearProperty = neopreneRubber.GetProperty(Name="Shear Test Data")
curveFit.AddTestData(TestData=shearProperty)
volumetricProperty = neopreneRubber.GetProperty(Name="Volumetric Test Data")
curveFit.AddTestData(TestData=volumetricProperty)
curveFit.RemoveTestData(TestData=uniaxialProperty)
curveFit.AddTestData(TestData=uniaxialProperty)
curveFit.Solve()
curveFit.CopyCoefficients(Destination=neoHookeanPropertyData)
curveFit.Delete()
```

**Delete**

Deletes a curve fitting.

**Example**

The following example loads a material with experimental test data and a Neo-Hookean hyperelastic property.

```python
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem()
GetChartData

Returns a generated graph data for the specified source data.

Valid source data are:

- Material Property
- Material Property Data

**Return**  The graph data for the specified source data.

**Type**  DataReference

RemoveTestData

Removes test data from a given curve fitting.

**Required Arguments**

**TestData**  The test data property to add.

**Type**  DataReference

**Example**

The following example loads a material with experimental test data and a Neo-Hookean hyperelastic property.

```python
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem()
engineeringData = system.GetContainer(ComponentName="Engineering Data")
neopreneRubber = engineeringData.ReadMaterial(
    Name="Neoprene Rubber",
    Source="Hyperelastic_Materials.xml")
neoHookeanProperty = neopreneRubber.GetProperty(Name="Neo-Hookean")
neoHookeanPropertyData = neoHookeanProperty.GetPropertyData(Name="Neo-Hookean")
curveFit = neoHookeanPropertyData.CreateCurveFitting(
    Type="Neo-Hookean",
    Definition="")
uniaxialProperty = neopreneRubber.GetProperty(Name="Uniaxial Test Data")
curveFit.AddTestData(TestData=uniaxialProperty)
biaxialProperty = neopreneRubber.GetProperty(Name="Biaxial Test Data")
curveFit.AddTestData(TestData=biaxialProperty)
shearProperty = neopreneRubber.GetProperty(Name="Shear Test Data")
curveFit.AddTestData(TestData=shearProperty)
volumetricProperty = neopreneRubber.GetProperty(Name="Volumetric Test Data")
curveFit.AddTestData(TestData=volumetricProperty)
curveFit.RemoveTestData(TestData=uniaxialProperty)
curveFit.AddTestData(TestData=uniaxialProperty)
curveFit.Solve()
curveFit.CopyCoefficients(Destination=neoHookeanPropertyData)
curveFit.Delete()
```
curveFit.AddTestData(TestData=uniaxialProperty)
biaxialProperty = neopreneRubber.GetProperty(Name="Biaxial Test Data")
curveFit.AddTestData(TestData=biaxialProperty)
shearProperty = neopreneRubber.GetProperty(Name="Shear Test Data")
curveFit.AddTestData(TestData=shearProperty)
volumetricProperty = neopreneRubber.GetProperty(Name="Volumetric Test Data")
curveFit.AddTestData(TestData=volumetricProperty)
curveFit.RemoveTestData(TestData=uniaxialProperty)
curveFit.AddTestData(TestData=uniaxialProperty)
curveFit.Solve()
curveFit.CopyCoefficients(Destination=neoHookanPropertyData)
curveFit.Delete()

**Solve**

The curve fitting module will solve for the coefficients which most nearly approximate the experimental test data.

**Engineering Data Favorite Items**

The container used by Engineering Data to maintain favorite items.

**Methods**

**GetMaterial**

Returns a material of a given name from a container.

The name matching is case insensitive. If a material is not found an exception is thrown.

**Return**  The material that matches the specified name.

**Type**  DataReference

**Required Arguments**

**Name**  The name of the material. This argument is case insensitive.

**Type**  string

**Example**

The following example creates a new Engineering Data system and queries the default material, Structural Steel.

```python
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem(Position="Default")
container = system.GetContainer(ComponentName="Engineering Data")
structuralSteel = container.GetMaterial(Name="Structural Steel")
```

**GetMaterials**

Returns the list of materials in a container. If no materials are in the container, the list is empty.

**Return**  A DataReferenceSet of the materials in the container.
Type  DataReferenceSet

**Load**

Populates the favorite items container with the user's favorite items.

Returns the favorite items container.

**Return**  The favorite items container.

Type  DataContainerReference

**Example**

The following example creates a new Engineering Data system and queries the default material, Structural Steel.

```python
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem(Position="Default")
favoritesContainer = EngData.LoadFavoriteItems()
structuralSteel = favoritesContainer.GetMaterial(Name="Structural Steel")
```

**Data Entities**

**Material**

The entity to store material information.

**Properties**

**Description**

The description of the material.

Type  string

Read Only  No

**DisplayName**

The display name of the material.

Type  string

Read Only  No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type  string

Read Only  No
**Source**

The path of the material’s source file ("*.engd" or "*.xml").

**Type** string

**Read Only** Yes

**Methods**

**AddToFavorites**

Adds an item to the Favorites item list.

**CreateProperty**

Includes a physical quantity or the constitutive relation for the physical response of a material. A property can be visualized as one or more tables of data made up of one or more dependent and independent variables.

The material property is created based on the specified optional parameters "Definition" and "Behavior".

**Return** Created material properly.

  **Type** DataReference

**Required Arguments**

**Name** The new material property name.

  **Type** string

**Optional Arguments**

**Behavior** The optional string to identify the way in which a new material property will behave.

  Behavior of some material properties can be specified in different ways, e.g., Elasticity can be specified as Isotropic, Orthotropic or Anisotropic.

  **Type** string

**Definition** The optional string to identify the way in which new material property will be defined.

  Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

  **Type** string

**Qualifiers** The optional dictionary of a qualifier name and it's corresponding value.

  **Type** Dictionary<string, string>

**Example**

The following example creates a new Engineering Data system and adds Coefficient of Thermal Expansion and Elasticity material properties to the default material Structural Steel.
Coefficient of Thermal Expansion is created using Secant definition and Orthotropic behavior. Elasticity is created using Orthotropic behavior.

```python
ENGDTemplate = GetTemplate(TemplateName="EngData")
ENGDSystem = ENGDTemplate.CreateSystem(Position="Default")

ENGDContainer = ENGDSystem.GetContainer(ComponentName="Engineering Data")
StructSteel = ENGDContainer.GetMaterial(Name="Structural Steel")

# Create material properties
CTEProperty = StructSteel.CreateProperty(
    Name="Coefficient of Thermal Expansion",
    Definition="Secant",
    Behavior="Orthotropic")
OrthoElaProperty = StructSteel.CreateProperty(
    Name="Elasticity",
    Behavior="Orthotropic")
```

**Delete**

Deletes the material.

**Duplicate**

Duplicates the data in this material and returns a new material. The name of the new material will be appended with a numerical value to make it unique.

**Return**

The material data reference of the duplicated material.

**Type**  
DataReference

**Required Arguments**

**TargetContainer**

The duplicated material will be added to this container.

**Type**  
DataContainerReference

**GetProperty**

Returns a material property of a given name from the specified material.

**Return**

The material property that matches the specified name.

**Type**  
DataReference

**Required Arguments**

**Name**

The name of the material property.

**Type**  
string

**Example**

The following example creates new Engineering Data system and queries Coefficient of Thermal Expansion property of the default material Structural Steel.

```python
ENGDTemplate = GetTemplate(TemplateName="EngData")
ENGDSystem = ENGDTemplate.CreateSystem(Position="Default")
```
ENGDContainer = ENGDSystem.GetContainer(ComponentName="Engineering Data")
StructSteel = ENGDContainer.GetMaterial(Name="Structural Steel")

# Get material property
CTEProperty = StructSteel.GetProperty(Name="Coefficient of Thermal Expansion")

**IsSuppressed**

Checks if an entity is suppressed.

Valid entities are:
- Material
- Material Property
- Material Property Data

Return Returns true if the entity is suppressed, false otherwise.

*Type* `bool`

**IsValid**

Validates a material data and provides a message in case of invalid data.

Return The flag that indicates if the material is valid.

*Type* `bool`

**Optional Arguments**

**Message** The validation failure message.

*Type* `Output<string>`

**Refresh**

This will repopulate the contents of a material with data from a given source. The NameInSource property of a Material will be used to find a match in the source to pull data from. In the event that NameInSource is not set, the DisplayName property will be used instead.

Note: This operation will cause destruction of any data currently in the material.

```
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem(Position="Default")
container = system.GetContainer(ComponentName="Engineering Data")
structSteel = container.GetMaterial(Name="Structural Steel")
density = structSteel.GetProperty(Name="Density")
density.Delete()
structSteel.UpdateMaterial(Source="General_Materials.xml")
restoredDensity = structSteel.GetProperty(Name="Density")
restoredDensity.SetData(SheetName="", Index=0, Variables=["Density"], Values=["8000 [kg m^-3]"])```

**Required Arguments**

**Source** The path of the source file.

*Type* `string`
**RemoveFromFavorites**

Removes an item from the Favorite items list.

**Optional Arguments**

- **Format**  
  The format of the file that contains the item.  
  
  **Type**  
  string

- **Name**  
  The name of the item to delete.  
  
  **Type**  
  string

- **Source**  
  The location of the file that contains the item on disk.  
  
  **Type**  
  string

- **Type**  
  The type of the object to delete. This is either a Material or a Mixture.  
  
  **Type**  
  EngineeringDataType

**Example**

The following example gets the list of favorites from Engineering Data. It then selects and deletes the "Gray Cast Iron" material from the list.

```plaintext
favorites = EngData.LoadFavoriteItems()  
matl = favorites.GetMaterial(Name="Gray Cast Iron")  
EngData.DeleteFromFavorites(
    Material=matl,
    Type="Material")
```

**SetAsDefaultFluidForModel**

This will specify the material to use (or not use) for parts in the model which are marked as a fluid.

If the material is in the Engineering Data component it will set or unset the material to be used in the model component of the system(s) that contain this Engineering Data component.

If the material is contained in Favorites it will set or unset the material to use as the default on fluids in the Engineering Data component when a new system is added to the project.

The material is set as the default in Favorites. It will automatically be added to the list of project defaults.

**Optional Arguments**

- **Default**  
  This Boolean is used to set or unset the material as the default.  
  
  **Type**  
  bool

  **Default Value**  
  True

**SetAsDefaultSolidForModel**

This will specify the material to use (or not use) for parts in the model which are marked as a solid.
If the material is in the Engineering Data component it will set or unset the material to be used in the model component of the system(s) that contain this Engineering Data component.

If the material is contained in Favorites it will set or unset the material to use as the default on solids in the Engineering Data component when a new system is added to the project.

The material is set as the default in Favorites. It will automatically be added to the list of project defaults.

**Optional Arguments**

**Default**

This Boolean is used to set or unset the material as the default.

- **Type** bool
- **Default Value** True

**SetSuppression**

Suppresses or Unsuppresses item.

Item can be suppressed to prevent it from being sent to a downstream cell in the system.

Following items can be suppressed:

Material

Material property

Material property data

**Required Arguments**

**Suppressed**

The flag to specify if the item should be suppressed or unsuppressed.

- **Type** bool

**Unlink**

Unlinks a material from its underlying source. Once this occurs, the material can no longer be restored to its original state and the material will no longer have a source.

**MaterialProperty**

The entity to store material property information.

A material property is the identifier for the singular information (for example, Density) that together with other properties defines or models the behavior of the material. A property is always defined by at least one table (tabular data), which could be singular. Some properties can contain a collection of tabular data (for example, Isotropic Elasticity).

**Properties**

**Behavior**

The string that defines the way in which the material property behaves, e.g., Elasticity has Isotropic, Orthotropic or Anisotropic behavior.
**Definition**

The definition of the material property. Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

**Description**

The description of the material property.

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**PropertyDataColl**

The collection of tabular data that defines the material property.

**TypeName**

The name of the material property.

**Methods**

**AddTestData**

Includes experimental test data for response function calculations.

**Required Arguments**

**TestData** The test data property to add.
BeginBatchUpdate

Marks the start of a series of data modifications to a table of data, to improve performance.

CreatePropertyData

Include an additional property data for a material property. Some properties may have more than one property data to describe the material property.

The preferred method of adding a material property data is to use CreateMaterialProperty.

Return  The new material properly data that was created.

Required Arguments

Name  The new material property data name.

Type  string

Optional Arguments

Behavior  A string to identify how the new material property data will behave.

The behavior of some material properties can be specified in different ways, e.g., Elasticity can be specified as Isotropic, Orthotropic or Anisotropic.

Type  string

Definition  A string to identify how the new material property data will be defined.

Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

Type  string

Qualifiers  The optional dictionary of a qualifier name and it's corresponding value.

Type  Dictionary<string, string>

Example

The following example creates the material property data Orthotropic Secant Coefficient of Thermal Expansion on the material property Coefficient of Thermal Expansion. This example assumes the material Structural Steel has been obtained from the General Materials library.

Get the material property we are going to create a new material property data on.

```
thermExpansionMatProp = structuralSteel.GetMaterialProperty(Name="Coefficient of Thermal Expansion")
```

Create the new material property data.
orthoSecantThermExpansionMatPropData = thermExpansionMatProp.CreatePropertyData(
    Name="Coefficient of Thermal Expansion",
    Definition="Secant",
    Behavior="Orthotropic")

**Delete**

Deletes the material property.

**Optional Arguments**

**Behavior**  The optional string to specify the material property behavior.

Behavior of some material properties can be specified in different ways e.g. Elasticity can be specified as Isotropic, Orthotropic or Anisotropic.

*Type*  string

**Definition**  The optional string to specify the material property definition.

Some material properties are defined in different ways e.g. Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

*Type*  string

**DeleteData**

Delete a row from a tabular data sheet.

**Required Arguments**

**Index**  Index of the row to delete.

*Type*  int

**Optional Arguments**

**SheetName**  Name of the sheet to access.

*Type*  string

**SheetQualifiers**  SheetQualifiers is used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of the Qualifier and its Value.

*Type*  Dictionary<string, string>

**Example**

The following example illustrates the deletion of a row from a tabular data sheet.

```csharp
# Create a new Engineering Data System and access Structural Steel
#
template1 = GetTemplate(TemplateName="EngData")
system1 = template1.CreateSystem()
engineeringData1 = system1.GetContainer(ComponentName="Engineering Data")
mat11 = engineeringData1.GetMaterial(Name="Structural Steel")
```
Delete the first row in the Density property

```matlab
matlProp1 = matl1.GetProperty(Name="Density")
matlProp1.DeleteTabularDataRow(Index = 0)
```

Delete the first row in the Coefficient of Thermal Expansion property with optional SheetName and SheetQualifiers

```matlab
matlProp2 = matl1.GetProperty(Name="Coefficient of Thermal Expansion")
matlProp2.DeleteTabularDataRow(
    SheetName="Coefficient of Thermal Expansion",
    SheetQualifiers={"Definition Method": "Secant", "Behavior": "Isotropic"},
    Index = 0)
```

**EndBatchUpdate**

Marks the completion of a series of data modifications.

**GetChartData**

Returns a generated graph data for the specified source data.

Valid source data are:

- Material Property
- Material Property Data

**Return** The graph data for the specified source data.

**Type** DataReference

**GetData**

Returns the tabular data associated with the data entity.

**Return** The returned data in scalar, list, or dictionary format.

**Type** Object

**Optional Arguments**

**AsDictionary** If set to true, the data will be returned as a dictionary where the keys are variable names and the values are the data for each variable. If set to false, the data will be returned in scalar or list format without the variable names.

**Type** bool

**Default Value** False

**ColumnMajor** If set to true, the data will be returned in column-major order. If set to false, the data will be returned in row-major order.

**Type** bool

**Default Value** True

**EndIndex** The end index for requesting a subset of the data (zero-based).
**SheetName**
Specifies the sheet name when the data contains multiple sheets.

**SheetQualifiers**
Used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of qualifiers and values.

**StartIndex**
The start index for requesting a subset of the data (zero-based).

**Variables**
Names of the variables for which data is requested (string or list of strings).

**Example**
In this example, all data is requested for the given tabular data entity.

```
 tabData1.GetData()
```

In this example, all data is requested in row-major order.

```
 tabData1.GetData(ColumnMajor=False)
```

In this example, all data is requested in dictionary format.

```
 tabData1.GetData(AsDictionary=True)
```

In this example, data for variables Density and Temperature is requested in dictionary format.

```
 tabData1.GetData(Variables="["Density", "Temperature"], AsDictionary=True)
```

**GetPropertyData**

Returns a property data of the specified material property.

The property data returned is based on specified optional parameters "Definition" and "Behavior".

**Return**
The material property data that matches specified type name, definition and behavior.

**Type**

DataReference
Required Arguments

Name   The material property data type name.
  Type   string

Optional Arguments

Behavior   The optional string to specify the material property data behavior.

  Behavior of some material properties can be specified in different ways, e.g., Elasticity can be specified as Isotropic, Orthotropic or Anisotropic.
  Type   string

Definition   The optional string to specify the material property data definition.

  Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.
  Type   string

Qualifiers   The optional dictionary of a qualifier name and it's corresponding value.
  Type   Dictionary<string, string>

Example

The following example creates new Engineering Data system and queries Coefficient of Thermal Expansion property data of the default material Structural Steel.

```csharp
ENGDTemplate = GetTemplate(TemplateName="EngData")
ENGDSystem = ENGDTemplate.CreateSystem(Position="Default")

ENGDContainer = ENGDSystem.GetContainer(ComponentName="Engineering Data")
StructSteel = ENGDContainer.GetMaterial(Name="Structural Steel")

# Get material property
CTEProperty = StructSteel.GetProperty(Name="Coefficient of Thermal Expansion")
# Get material property data
CTEPropData = CTEProperty.GetPropertyData(
   Name="Coefficient of Thermal Expansion",
   Definition="Secant",
   Behavior="Isotropic")
RefTempPropData = CTEProperty.GetPropertyData(
   Name="Reference Temperature",
   Definition="Secant",
   Behavior="Isotropic")
```

IsSuppressed

Checks if an entity is suppressed.

Valid entities are:

  Material
  Material Property
  Material Property Data
**Return** Returns true if the entity is suppressed, false otherwise.

**Type** bool

**IsValid**

Validates a material property and provides a message in case of invalid data.

**Return** The flag that indicates if the material property is valid.

**Type** bool

**Required Arguments**

**Material** The parent material of the property.

**Type** DataReference

**Optional Arguments**

**Message** The validation failure message.

**Type** Output<string>

**RemoveTestData**

Excludes experimental test data for response function calculations.

**Required Arguments**

**TestData** The test data property to add.

**Type** DataReference

**SetData**

Set tabular data associated with the data entity.

**Optional Arguments**

**Data** Sets the data using a dictionary form. The keys are the variable names and the values are the data. The use of this argument is mutually exclusive with "Values" and "Variables".

**Type** Dictionary<string, List<Object>>

**Index** Specifies the starting location used to set the data (zero-based). A value of -1 indicates that the data should be appended to the existing data.

**Type** int

**Default Value** 0

**SheetName** Specifies the sheet name when the data contains multiple sheets.

**Type** string
SheetQualifiers

Used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of qualifiers and values.

Type  Dictionary<string, string>

Values

List of data values set in conjunction with the "Variables" parameter. This parameter and the "Data" parameter are mutually exclusive.

Type  List<List<Object>>

Variables

Names of the variables for which data is being set. This parameter and the "Data" parameter are mutually exclusive.

Type  List<string>

Example

```csharp
// Create a new Engineering Data System and access Structural Steel

template1 = GetTemplate(TemplateName="EngData")
system1 = template1.CreateSystem()
engineeringData1 = system1.GetContainer(ComponentName="Engineering Data")
matl1 = engineeringData1.GetMaterial(Name="Structural Steel")

// Change the value of a simple single-valued property
matlProp1 = matl1.GetProperty(Name="Density")
matlProp1.SetData(
  Variables="Density",
  Values="8500 [kg m^-3]"
)

// Set Temperature-dependent data for Elasticity based on lists of variables and values.
matlProp2 = matl1.GetProperty(Name="Elasticity")
temperature = ["400 [K]", "600 [K]", "800 [K]"
E = ["2e5 [MPa]", "1.9e5 [MPa]", "1.6e5 [MPa]"
matlProp2.SetData(
  Variables = ["Temperature","Young's Modulus"],
  Values = [temperature, E]
)

// Change the Temperature for the second table entry.
matlProp2.SetData(
  Index = 1,
  Variables = "Temperature",
  Values = "625 [K]"
)

// Set a list for Poisson's Ratio starting at the second table entry.
matlProp2.SetData(
  Index = 1,
  Variables = "Poisson's Ratio",
  Values = [0.3, 0.3]
)

// Set Temperature-dependent property data for the Coefficient of Thermal Expansion using a dictionary. The dictionary key is the Variable name, followed by the list of values for the variable.
matlProp3 = matl1.GetProperty(Name="Coefficient of Thermal Expansion")
newData = {"Temperature": ["200 [F]","400 [F]","600 [F]","800 [F]","1000 [F]"],
"Coefficient of Thermal Expansion" : ["6.3e-6 [F^-1]", "7.0e-6 [F^-1]",
"7.4e-6 [F^-1]", "7.8e-6 [F^-1]",
"8.04e-6 [F^-1]"]}
matlProp3.SetData(
  SheetName="Coefficient of Thermal Expansion",
  SheetQualifiers={"Definition Method": "Secant", "Behavior": "Isotropic"},
)"
**SetQualifier**

Changes the values of a specific qualifier in a data table.

**Required Arguments**

- **Qualifier**
  - The Qualifier to Set.
  - **Type** string

- **Value**
  - The new value.
  - **Type** string

**Optional Arguments**

- **SheetName**
  - The name of the tabular data sheet that contains the qualifier.
  - **Type** string

- **SheetQualifiers**
  - SheetQualifiers can be used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of the Qualifier and its Value.
  - **Type** Dictionary<string, string>

- **VariableName**
  - The name of the Variable that contains the qualifier to be changed.
  - **Type** string

- **VariableQualifiers**
  - VariableQualifiers can be used to pass in the qualifiers to select between multiple variables with the same name. This is a dictionary of the Qualifier and its Value.
  - **Type** Dictionary<string, string>

**Example**

The following example changes the 'Derive From' setting within an Isotropic Elasticity material property to be "Bulk Modulus and Poisson's Ratio".

```csharp
mat1 = engineeringData1.GetMaterial(Name="Structural Steel")
mat1Prop1 = mat1.GetProperty(Name="Elasticity")
mat1Prop1.SetQualifier(
    SheetName="Elasticity",
    Qualifier="Derive from",
    Value="Bulk Modulus and Poisson's Ratio")
```

**SetSuppression**

Suppresses or Unsuppresses item.

Item can be suppressed to prevent it from being sent to a downstream cell in the system.

Following items can be suppressed:

- Material
Material property

Material property data

**Required Arguments**

**Suppresses** The flag to specify if the item should be suppressed or unsuppressed.

**MaterialPropertyData**

The entity to store material property (tabular) data information. The material property data is a collection of material variable data.

**Properties**

**Behavior**

The behavior of the material variable tabular data. Some material properties can have different behavior, e.g., Elasticity has Isotropic, Orthotropic or Anisotropic behavior.

**Definition**

The definition of the material variable tabular data. Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

**DependentColl**

The collection of dependent variables in the material variable tabular data, e.g., Density.

**Description**

The description of the material variable tabular data.

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.
Engineering Data

**Type** string

**Read Only** No

**IndependentColl**

The collection of independent variables in the material variable tabular data, e.g., Temperature.

**Type** DataReferenceSet

**Read Only** Yes

**PrimaryIndependent**

The primary independent variable in the material variable tabular data, e.g., Temperature.

**Type** DataReference

**Read Only** Yes

**RowsCount**

The number of data values for a variable in the material variable tabular data.

**Type** int

**Read Only** Yes

**TypeName**

The name of the material variable tabular data.

**Type** string

**Read Only** No

**VariableColl**

The collection of variables in the material variable tabular data.

**Type** DataReferenceSet

**Read Only** Yes

**Methods**

**BeginBatchUpdate**

Marks the start of a series of data modifications to a table of data, to improve performance.

**CreateCurveFitting**

Creates a curve fitting for a given property data.

**Return** The curve fitting.
Type  DataReference

Required Arguments

**Definition**  The definition of curve fitting to create. This must be a definition that is supported by an engineering data curve fitting.

i.e. 1 Parameter, 2 Parameter, 1st Order, 2nd Order

**Type**  string

**Type**  The type of curve fitting to create. This must be a type that is supported by an engineering data curve fitting.

i.e. Neo-Hookean, Mooney-Rivlin, Ogden, Yeoh, Polynomial

**Type**  string

Example

The following example loads a material with experimental test data and a Neo-Hookean hyperelastic property.

```python
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem()
engineeringData = system.GetContainer(ComponentName="Engineering Data")
neopreneRubber = engineeringData.ReadMaterial(
    Name="Neoprene Rubber",
    Source="Hyperelastic_Materials.xml")
neoHookeanProperty = neopreneRubber.GetProperty(Name="Neo-Hookean")
neoHookeanPropertyData = neoHookeanProperty.GetPropertyData(Name="Neo-Hookean")
curveFit = neoHookeanPropertyData.CreateCurveFitting(
    Type="Neo-Hookean",
    Definition="")
uniaxialProperty = neopreneRubber.GetProperty(Name="Uniaxial Test Data")
curveFit.AddTestData(TestData=uniaxialProperty)
biaxialProperty = neopreneRubber.GetProperty(Name="Biaxial Test Data")
curveFit.AddTestData(TestData=biaxialProperty)
shearProperty = neopreneRubber.GetProperty(Name="Shear Test Data")
curveFit.AddTestData(TestData=shearProperty)
volumetricProperty = neopreneRubber.GetProperty(Name="Volumetric Test Data")
curveFit.AddTestData(TestData=volumetricProperty)
curveFit.RemoveTestData(TestData=uniaxialProperty)
curveFit.AddTestData(TestData=uniaxialProperty)
curveFit.Solve()
curveFit.CopyCoefficients(Destination=neoHookeanPropertyData)
curveFit.Delete()
```

**CreateVariable**

Include an additional variable in the property data for a material property.

**Return**  The new variable data that was created.

**Type**  DataReference

Required Arguments

**Name**  The new variable data name.

**Type**  string
Optional Arguments

Qualifiers  The optional dictionary of a qualifier name and it's corresponding value.

Type  Dictionary<string, string>

Delete

Deletes the material property data.

EndBatchUpdate

Marks the completion of a series of data modifications.

GetChartData

Returns a generated graph data for the specified source data.

Valid source data are:

- Material Property
- Material Property Data

Return  The graph data for the specified source data.

Type  DataReference

GetCurveFitting

Query to return the DataReference to the CurveFit used by some PropertyData.

Return  CurveFit DataReference

Type  DataReference

GetVariable

Returns a material variable of a given name from a material property data.

Return  The requested variable.

Type  DataReference

Required Arguments

Name  The name of the variable.

Type  string

Example

The following example creates new Engineering Data system and queries Coefficient of Thermal Expansion property data of the default material Structural Steel.

```
ENGDTemplate = GetTemplate(TemplateName="EngData")
ENGDSystem = ENGDTemplate.CreateSystem(Position="Default")
```
ENGDContainer = ENGDSystem.GetContainer(ComponentName="Engineering Data")
StructSteel = ENGDContainer.GetMaterial(Name="Structural Steel")

# Get material
CTEProperty = StructSteel.GetProperty(Name="Coefficient of Thermal Expansion")
# Get material property data
CTEPropData = CTEProperty.GetPropertyData(  
    Name="Coefficient of Thermal Expansion",  
    Definition="Secant",  
    Behavior="Isotropic")
# Get material variable
CTEVariable = CTEPropData.GetVariable(  
    Name="Coefficient of Thermal Expansion")

**IsSuppressed**

Checks if an entity is suppressed.

Valid entities are:

- Material
- Material Property
- Material Property Data

**Return** Returns true if the entity is suppressed, false otherwise.

**Type** bool

**IsValid**

Validates a material property data and provides a message in case of invalid data.

**Return** The flag that indicates if the material property data is valid.

**Type** bool

**Required Arguments**

**Material** The parent material of the material property data.

**Type** DataReference

**Optional Arguments**

**Message** The validation failure message.

**Type** Output<string>

**SetSuppression**

Suppresses or Unsuppresses item.

Item can be suppressed to prevent it from being sent to a downstream cell in the system.

Following items can be suppressed:

Material

Material property
Material property data

**Required Arguments**

**Suppressed**  The flag to specify if the item should be suppressed or unsuppressed.

  **Type**  bool

---

**Engineering Data Favorite Library**

The container used by Engineering Data to allow working with a library.

**Methods**

**Close**

Closes an Engineering Data library.

**CreateMaterial**

Adds a new material to the container.

  **Return**  The material data reference of the new material.

  **Type**  DataReference

**Required Arguments**

**Name**  The name to be used for this material.

  **Type**  string

**Export**

Exports engineering data to the specified file.

The following type of file format is supported for export:

  MatML 3.1 schema for Material(s)

**Required Arguments**

**FilePath**  The target path for the Engineering Data file ("*.xml").

  **Type**  string

**Format**  The file format in which engineering data will be exported.

  **Type**  string

**UnitSystem**  The unit system in which engineering data will be exported.

  **Type**  string
Optional Arguments

**ApplyScaleOffset**
- The flag to specify if the scale factor and offset value will be applied during export.
- **Type**: bool
- **Default Value**: False

**IgnoreSuppressed**
- The flag to specify if suppressed engineering data will be ignored during export.
- **Type**: bool
- **Default Value**: False

**OverwriteTarget**
- The flag to specify if the target file will be overwritten.
- **Type**: bool
- **Default Value**: False

**ReplaceMaterial**
- The flag to specify if the earlier material data will be replaced in case of similar material names.
- **Type**: bool
- **Default Value**: False

**Example**

```plaintext
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem(Position="Default")
container = system.GetContainer(ComponentName="Engineering Data")
```

**GetMaterial**

Returns a material of a given name from a container.

The name matching is case insensitive. If a material is not found an exception is thrown.

**Return**
- The material that matches the specified name.

  **Type**: DataReference

**Required Arguments**

**Name**
- The name of the material. This argument is case insensitive.

  **Type**: string

**Example**

The following example creates a new Engineering Data system and queries the default material, Structural Steel.

```plaintext
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem(Position="Default")
container = system.GetContainer(ComponentName="Engineering Data")
```
structuralSteel = container.GetMaterial(Name="Structural Steel")

**GetMaterials**

Returns the list of materials in a container. If no materials are in the container, the list is empty.

*Return*  
A DataReferenceSet of the materials in the container.

*Type*  
DataReferenceSet

**Import**

Imports engineering data into an existing source from a specified source.

The following types of files are supported for import:

- Engineering Data libraries exported from Workbench 9.0 to 11.0 SP1
- Material(s) file following the MatML 3.1 schema
- Material(s) file generated by AUTODYN

**Required Arguments**

**Source**  
The source which contains the materials.

*Type*  
string

**ImportMaterial**

Reads the data for a single material into the requested container.

*Return*  
A DataReference for the material that was read.

*Type*  
DataReference

**Required Arguments**

**Name**  
The name of the material to read from the source.

*Type*  
string

**Source**  
The source which contains the requested material.

*Type*  
string

**Example**

This code shows how to read the Copper Alloy material from the provided samples, into Engineering Data to use in an analysis.

```python/installDir = r"C:\Program Files\ANSYS Inc\v121"  /mat11 = engineeringData1.ReadMaterial(  
    Name="Copper Alloy",  
    Source=installDir+r"\Addins\EngineeringData\Samples\General_Materials.xml")```
**Refresh**

This will repopulate the contents of an Engineering Data library with data from its source.

**Save**

Saves an Engineering Data library.

**Optional Arguments**

**FilePath**  
The optional string to specify the path if the Engineering Data library should be saved at different location.

  - **Type**  
    - string

---

**Data Entities**

**Material**

The entity to store material information.

**Properties**

**Description**

The description of the material.

  - **Type**  
    - string
  - **Read Only**  
    - No

**DisplayName**

The display name of the material.

  - **Type**  
    - string
  - **Read Only**  
    - No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

  - **Type**  
    - string
  - **Read Only**  
    - No

**Source**

The path of the material's source file ("*.engd" or "*.xml").

  - **Type**  
    - string
  - **Read Only**  
    - Yes
Methods

AddToFavorites

Adds an item to the Favorites item list.

CreateProperty

Includes a physical quantity or the constitutive relation for the physical response of a material. A property can be visualized as one or more tables of data made up of one or more dependent and independent variables.

The material property is created based on the specified optional parameters "Definition" and "Behavior".

Return  Created material properly.

Type  DataReference

Required Arguments

Name  The new material property name.

Type  string

Optional Arguments

Behavior  The optional string to identify the way in which a new material property will behave.

Behavior of some material properties can be specified in different ways, e.g., Elasticity can be specified as Isotropic, Orthotropic or Anisotropic.

Type  string

Definition  The optional string to identify the way in which new material property will be defined.

Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

Type  string

Qualifiers  The optional dictionary of a qualifier name and it's corresponding value.

Type  Dictionary<string, string>

Example

The following example creates a new Engineering Data system and adds Coefficient of Thermal Expansion and Elasticity material properties to the default material Structural Steel.

Coefficient of Thermal Expansion is created using Secant definition and Orthotropic behavior. Elasticity is created using Orthotropic behavior.

    ENGDTemplate = GetTemplate(TemplateName="EngData")
    ENGDSystem = ENGDTemplate.CreateSystem(Position="Default")
    ENGDContainer = ENGDSystem.GetContainer(ComponentName="Engineering Data")
StructSteel = ENGDContainer.GetMaterial(Name="Structural Steel")

Create material properties
CTEProperty = StructSteel.CreateProperty(
    Name="Coefficient of Thermal Expansion",
    Definition="Secant",
    Behavior="Orthotropic")
OrthoElasProperty = StructSteel.CreateProperty(
    Name="Elasticity",
    Behavior="Orthotropic")

Delete
Delete the material.

Duplicate
Duplicates the data in this material and returns a new material. The name of the new material will be appended with a numerical value to make it unique.

Return
The material data reference of the duplicated material.

Type
DataReference

Required Arguments
TargetContainer
The duplicated material will be added to this container.

Type
DataContainerReference

GetProperty
Returns a material property of a given name from the specified material.

Return
The material property that matches the specified name.

Type
DataReference

Required Arguments
Name
The name of the material property.

Type
string

Example
The following example creates new Engineering Data system and queries Coefficient of Thermal Expansion property of the default material Structural Steel.

ENGDTemplate = GetTemplate(TemplateName="EngData")
ENGDSystem = ENGDTemplate.CreateSystem(Position="Default")
ENGDContainer = ENGDSystem.GetContainer(ComponentName="Engineering Data")
StructSteel = ENGDContainer.GetMaterial(Name="Structural Steel")

# Get material property
CTEProperty = StructSteel.GetProperty(Name="Coefficient of Thermal Expansion")
**IsSuppressed**

Checks if an entity is suppressed.

Valid entities are:

- Material
- Material Property
- Material Property Data

**Return**  
Returns true if the entity is suppressed, false otherwise.

**Type**  
bool

**IsValid**

Validates a material data and provides a message in case of invalid data.

**Return**  
The flag that indicates if the material is valid.

**Type**  
bool

**Optional Arguments**

**Message**  
The validation failure message.

**Type**  
Output<string>

**Refresh**

This will repopulate the contents of a material with data from a given source. The NameInSource property of a Material will be used to find a match in the source to pull data from. In the event that NameInSource is not set, the DisplayName property will be used instead.

Note: This operation will cause destruction of any data currently in the material.

```csharp
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem(Position="Default")
container = system.GetContainer(ComponentName="Engineering Data")
structSteel = container.GetMaterial(Name="Structural Steel")
density = structSteel.GetProperty(Name="Density")
density.Delete()
structSteel.UpdateMaterial(Source="General_Materials.xml")
restoredDensity = structSteel.GetProperty(Name="Density")
restoredDensity.SetData(SheetName="", Index=0, Variables=["Density"], Values=["8000 [kg m^-3]"])
```

**Required Arguments**

**Source**  
The path of the source file.

**Type**  
string

**RemoveFromFavorites**

Removes an item from the Favorite items list.
Optional Arguments

**Format**

The format of the file that contains the item.

*Type*  
*string*

**Name**

The name of the item to delete.

*Type*  
*string*

**Source**

The location of the file that contains the item on disk.

*Type*  
*string*

**Type**

The type of the object to delete. This is either a Material or a Mixture.

*Type*  
*EngineeringDataType*

**Example**

The following example gets the list of favorites from Engineering Data. It then selects and deletes the "Gray Cast Iron" material from the list.

```python
favorites = EngData.LoadFavoriteItems()
matl = favorites.GetMaterial(Name="Gray Cast Iron")
EngData.DeleteFromFavorites(
    Material=matl,
    Type="Material")
```

**SetAsDefaultFluidForModel**

This will specify the material to use (or not use) for parts in the model which are marked as a fluid.

If the material is in the Engineering Data component it will set or unset the material to be used in the model component of the system(s) that contain this Engineering Data component.

If the material is contained in Favorites it will set or unset the material to use as the default on fluids in the Engineering Data component when a new system is added to the project.

The material is set as the default in Favorites. It will automatically be added to the list of project defaults.

Optional Arguments

**Default**

This Boolean is used to set or unset the material as the default.

*Type*  
*bool*

*Default Value*  
*True*

**SetAsDefaultSolidForModel**

This will specify the material to use (or not use) for parts in the model which are marked as a solid.

If the material is in the Engineering Data component it will set or unset the material to be used in the model component of the system(s) that contain this Engineering Data component.
If the material is contained in Favorites it will set or unset the material to use as the default on solids in the Engineering Data component when a new system is added to the project.

The material is set as the default in Favorites. It will automatically be added to the list of project defaults.

Optional Arguments

**Default**  
This Boolean is used to set or unset the material as the default.

- **Type**: bool
- **Default Value**: True

**SetSuppression**

Suppresses or Unsuppresses item.

Item can be suppressed to prevent it from being sent to a downstream cell in the system.

Following items can be suppressed:

- Material
- Material property
- Material property data

Required Arguments

**Suppressed**  
The flag to specify if the item should be suppressed or unsuppressed.

- **Type**: bool

**Unlink**

Unlinks a material from its underlying source. Once this occurs, the material can no longer be restored to its original state and the material will no longer have a source.

**MaterialProperty**

The entity to store material property information.

A material property is the identifier for the singular information (for example, Density) that together with other properties defines or models the behavior of the material. A property is always defined by at least one table (tabular data), which could be singular. Some properties can contain a collection of tabular data (for example, Isotropic Elasticity).

**Properties**

**Behavior**

The string that defines the way in which the material property behaves, e.g., Elasticity has Isotropic, Orthotropic or Anisotropic behavior.

- **Type**: string
**Read Only**  No

**Definition**

The definition of the material property. Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

**Type**  string

**Description**

The description of the material property.

**Type**  string

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  string

**PropertyDataColl**

The collection of tabular data that defines the material property.

**Type**  DataReferenceSet

**TypeName**

The name of the material property.

**Type**  string

**Methods**

**AddTestData**

Includes experimental test data for response function calculations.

**Required Arguments**

**TestData**  The test data property to add.

**Type**  DataReference
**BeginBatchUpdate**

Marks the start of a series of data modifications to a table of data, to improve performance.

**CreatePropertyData**

Include an additional property data for a material property. Some properties may have more than one property data to describe the material property.

The preferred method of adding a material property data is to use CreateMaterialProperty.

**Return** The new material property data that was created.

**Type**  DataReference

**Required Arguments**

**Name**  The new material property data name.

**Type**  string

**Optional Arguments**

**Behavior**  A string to identify how the new material property data will behave.

The behavior of some material properties can be specified in different ways, e.g., Elasticity can be specified as Isotropic, Orthotropic or Anisotropic.

**Type**  string

**Definition**  A string to identify how the new material property data will be defined.

Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

**Type**  string

**Qualifiers**  The optional dictionary of a qualifier name and it's corresponding value.

**Type**  Dictionary<string, string>

**Example**

The following example creates the material property data Orthotropic Secant Coefficient of Thermal Expansion on the material property Coefficient of Thermal Expansion. This example assumes the material Structural Steel has been obtained from the General Materials library.

Get the material property we are going to create a new material property data on.

```csharp
thermExpansionMatProp = structuralSteel.GetMaterialProperty(Name="Coefficient of Thermal Expansion")
```

Create the new material property data.

```csharp
orthoSecantThermExpansionMatPropData = thermExpansionMatProp.CreatePropertyData(
    Name="Coefficient of Thermal Expansion",
);```
Definition="Secant",
Behavior="Orthotropic")

**Delete**

Deletes the material property.

**Optional Arguments**

**Behavior**  
The optional string to specify the material property behavior.

Behavior of some material properties can be specified in different ways e.g. Elasticity can be specified as Isotropic, Orthotropic or Anisotropic.

**Type**  
string

**Definition**  
The optional string to specify the material property definition.

Some material properties are defined in different ways e.g. Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

**Type**  
string

**DeleteData**

Delete a row from a tabular data sheet.

**Required Arguments**

**Index**  
Index of the row to delete.

**Type**  
int

**Optional Arguments**

**SheetName**  
Name of the sheet to access.

**Type**  
string

**SheetQualifiers**  
SheetQualifiers is used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of the Qualifier and its Value.

**Type**  
Dictionary<string, string>

**Example**

The following example illustrates the deletion of a row from a tabular data sheet.

```csharp
# Create a new Engineering Data System and access Structural Steel

template1 = GetTemplate(TemplateName="EngData")
system1 = template1.CreateSystem()
engineeringData1 = system1.GetContainer(ComponentName="Engineering Data")
matl1 = engineeringData1.GetMaterial(Name="Structural Steel")

# Delete the first row in the Density property
```
matlProp1 = matl1.GetProperty(Name="Density")
matlProp1.DeleteTabularDataRow(Index = 0)
#
# Delete the first row in the Coefficient of Thermal Expansion property with optional SheetNamea nd SheetQualifiers
#
matlProp2 = matl1.GetProperty(Name="Coefficient of Thermal Expansion")
matlProp2.DeleteTabularDataRow(
    SheetName="Coefficient of Thermal Expansion",
    SheetQualifiers={"Definition Method": "Secant", "Behavior": "Isotropic"},
    Index = 0)

EndBatchUpdate
Marks the completion of a series of data modifications.

GetChartData
Returns a generated graph data for the specified source data.
Valid source data are:

    Material Property
    Material Property Data

Return The graph data for the specified source data.
    Type DataReference

GetData
Returns the tabular data associated with the data entity.
Return The returned data in scalar, list, or dictionary format.
    Type Object

Optional Arguments

AsDictionary If set to true, the data will be returned as a dictionary where the keys are variable names and the values are the data for each variable. If set to false, the data will be returned in scalar or list format without the variable names.
    Type bool
    Default Value False

ColumnMajor If set to true, the data will be returned in column-major order. If set to false, the data will be returned in row-major order.
    Type bool
    Default Value True

EndIndex The end index for requesting a subset of the data (zero-based).
    Type int
    Default Value -2147483647
SheetName  Specifies the sheet name when the data contains multiple sheets.

Type    string

SheetQualifiers  Used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of qualifiers and values.

Type    Dictionary<string, string>

StartIndex  The start index for requesting a subset of the data (zero-based).

Type    int
Default Value    0

Variables  Names of the variables for which data is requested (string or list of strings).

Type    Object

Example
In this example, all data is requested for the given tabular data entity.

```csharp
    tabData1.GetData()
```

In this example, all data is requested in row-major order.

```csharp
    tabData1.GetData(ColumnMajor=False)
```

In this example, all data is requested in dictionary format.

```csharp
    tabData1.GetData(AsDictionary=True)
```

In this example, data for variables Density and Temperature is requested in dictionary format.

```csharp
    tabData1.GetData(Variables=["Density", "Temperature"], AsDictionary=True)
```

GetPropertyData
Returns a property data of the specified material property.

The property data returned is based on specified optional parameters "Definition" and "Behavior".

Return  The material property data that matches specified type name, definition and behavior.

Type    DataReference

Required Arguments

Name  The material property data type name.

Type    string
Optional Arguments

**Behavior**
The optional string to specify the material property data behavior.

Behavior of some material properties can be specified in different ways, e.g., Elasticity can be specified as Isotropic, Orthotropic or Anisotropic.

*Type* string

**Definition**
The optional string to specify the material property data definition.

Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

*Type* string

**Qualifiers**
The optional dictionary of a qualifier name and its corresponding value.

*Type* Dictionary<string, string>

**Example**
The following example creates a new Engineering Data system and queries Coefficient of Thermal Expansion property data of the default material Structural Steel.

```
ENGDTemplate = GetTemplate(TemplateName="EngData")
ENGDSystem = ENGDTemplate.CreateSystem(Position="Default")
ENGDContainer = ENGDSystem.GetContainer(ComponentName="Engineering Data")
StructSteel = ENGDContainer.GetMaterial(Name="Structural Steel")

# Get material property
CTEProperty = StructSteel.GetProperty(Name="Coefficient of Thermal Expansion")

# Get material property data
CTEPropData = CTEProperty.GetPropertyData(
    Name="Coefficient of Thermal Expansion",
    Definition="Secant",
    Behavior="Isotropic")
RefTempPropData = CTEProperty.GetPropertyData(
    Name="Reference Temperature",
    Definition="Secant",
    Behavior="Isotropic")
```

**IsSuppressed**
Checks if an entity is suppressed.

Valid entities are:

- Material
- Material Property
- Material Property Data

*Return* Returns true if the entity is suppressed, false otherwise.

*Type* bool
**IsValid**

Validates a material property and provides a message in case of invalid data.

**Return**  The flag that indicates if the material property is valid.

  **Type**  bool

**Required Arguments**

**Material**  The parent material of the property.

  **Type**  DataReference

**Optional Arguments**

**Message**  The validation failure message.

  **Type**  Output<string>

**RemoveTestData**

Excludes experimental test data for response function calculations.

**Required Arguments**

**TestData**  The test data property to add.

  **Type**  DataReference

**SetData**

Set tabular data associated with the data entity.

**Optional Arguments**

**Data**  
Sets the data using a dictionary form. The keys are the variable names and the values are the data. The use of this argument is mutually exclusive with "Values" and "Variables".

  **Type**  Dictionary<string, List<Object>>

**Index**  Specifies the starting location used to set the data (zero-based). A value of -1 indicates that the data should be appended to the existing data.

  **Type**  int

  **Default Value**  0

**SheetName**  Specifies the sheet name when the data contains multiple sheets.

  **Type**  string

**SheetQualifiers**  Used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of qualifiers and values.
**Type**  Dictionary\(<\text{string}, \text{string}>\)

**Values**  List of data values set in conjunction with the "Variables" parameter. This parameter and the "Data" parameter are mutually exclusive.

**Type**  List\(<\text{List}<>\text{Object}></text>\)

**Variables**  Names of the variables for which data is being set. This parameter and the "Data" parameter are mutually exclusive.

**Type**  List\(<\text{string}>\)

**Example**

```python
# Create a new Engineering Data System and access Structural Steel

system1 = template1.CreateSystem()

matl1 = system1.GetContainer(ComponentName="Engineering Data")

# Change the value of a simple single-valued property

matlProp1 = matl1.GetProperty(Name="Density")
matlProp1.SetData(
    Variables="Density",
    Values="8500 \ [kg \ m^{-3}]"
)

# Set Temperature-dependent data for Elasticity based on lists of variables and values.

matlProp2 = matl1.GetProperty(Name="Elasticity")
temperature = ["400 \ [K]", "600 \ [K]", "800 \ [K]"
E = ["2e5 \ [MPa]", "1.9e5 \ [MPa]", "1.6e5 \ [MPa]"
matlProp2.SetData(
    Variables = ["Temperature","Young's Modulus"],
    Values = [temperature, E]
)

# Change the Temperature for the second table entry.

matlProp2.SetData(
    Index = 1,
    Variables = "Temperature",
    Values = "625 \ [K]"
)

# Set a list for Poisson's Ratio starting at the second table entry.

matlProp2.SetData(
    Index = 1,
    Variables = "Poisson's Ratio",
    Values = [0.3, 0.3]
)

# Set Temperature-dependent property data for the Coefficient of Thermal Expansion using a dictionary. The dictionary key is the Variable name, followed by the list of values for the variable.

newData = {
    "Temperature": ["200 \ [F]", "400 \ [F]", "600 \ [F]", "800 \ [F]", "1000 \ [F]"],
    "Coefficient of Thermal Expansion": ["6.3e-6 \ [F^{-1}]", "7.0e-6 \ [F^{-1}]",
    "7.46e-6 \ [F^{-1}]", "7.8e-6 \ [F^{-1}]",
    "8.04e-6 \ [F^{-1}]"
}

matlProp3.SetData(
    SheetName="Coefficient of Thermal Expansion",
    SheetQualifiers={"Definition Method": "Secant", "Behavior": "Isotropic"},
    Data = newData
)
```


**SetQualifier**

Changes the values of a specified qualifier in a data table.

**Required Arguments**

**Qualifier**

The Qualifier to Set.

- **Type** string

**Value**

The new value.

- **Type** string

**Optional Arguments**

**SheetName**

The name of the tabular data sheet that contains the qualifier.

- **Type** string

**SheetQualifiers**

SheetQualifiers can be used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of the Qualifier and its Value.

- **Type** Dictionary<

**VariableName**

The name of the Variable that contains the qualifier to be changed.

- **Type** string

**VariableQualifiers**

VariableQualifiers can be used to pass in the qualifiers to select between multiple variables with the same name. This is a dictionary of the Qualifier and its Value.

- **Type** Dictionary<

**Example**

The following example changes the 'Derive From' setting within an Isotropic Elasticity material property to be "Bulk Modulus and Poisson's Ratio".

```python
mat1 = engineeringData1.GetMaterial(Name="Structural Steel")
matProp1 = mat1.GetProperty(Name="Elasticity")
matProp1.SetQualifier(
    SheetName="Elasticity",
    Qualifier="Derive from",
    Value="Bulk Modulus and Poisson's Ratio")
```

**SetSuppression**

Suppresses or Unsuppresses item.

Item can be suppressed to prevent it from being sent to a downstream cell in the system.

Following items can be suppressed:

- Material
- Material property
Material property data

**Required Arguments**

**Suppressed**  The flag to specify if the item should be suppressed or unsuppressed.

   Type  bool

**MaterialPropertyData**

The entity to store material property (tabular) data information. The material property data is a collection of material variable data.

**Properties**

**Behavior**

The behavior of the material variable tabular data. Some material properties can have different behavior, e.g., Elasticity has Isotropic, Orthotropic or Anisotropic behavior.

   Type  string
   Read Only  No

**Definition**

The definition of the material variable tabular data. Some material properties are defined in different ways, e.g., Thermal Expansion can be defined as Secant Coefficient of Thermal Expansion and Instantaneous Coefficient of Thermal Expansion.

   Type  string
   Read Only  No

**DependentColl**

The collection of dependent variables in the material variable tabular data, e.g., Density.

   Type  DataReferenceSet
   Read Only  Yes

**Description**

The description of the material variable tabular data.

   Type  string
   Read Only  No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

   Type  string
**Read Only**  No

**IndependentColl**
The collection of independent variables in the material variable tabular data, e.g., Temperature.

**Type**  DataReferenceSet

**Read Only**  Yes

**PrimaryIndependent**
The primary independent variable in the material variable tabular data, e.g., Temperature.

**Type**  DataReference

**Read Only**  Yes

**RowsCount**
The number of data values for a variable in the material variable tabular data.

**Type**  int

**Read Only**  Yes

**TypeName**
The name of the material variable tabular data.

**Type**  string

**Read Only**  No

**VariableColl**
The collection of variables in the material variable tabular data.

**Type**  DataReferenceSet

**Read Only**  Yes

**Methods**

**BeginBatchUpdate**
Marks the start of a series of data modifications to a table of data, to improve performance.

**CreateCurveFitting**
Creates a curve fitting for a given property data.

**Return**
The curve fitting.

**Type**  DataReference
Required Arguments

Definition  The definition of curve fitting to create. This must be a definition that is supported by an engineering data curve fitting.

i.e. 1 Parameter, 2 Parameter, 1st Order, 2nd Order

Type  string

Type  The type of curve fitting to create. This must be a type that is supported by an engineering data curve fitting.

i.e. Neo-Hookean, Mooney-Rivlin, Ogden, Yeoh, Polynomial

Type  string

Example
The following example loads a material with experimental test data and a Neo-Hookean hyperelastic property.

```python
template = GetTemplate(TemplateName="EngData")
system = template.CreateSystem()
engineeringData = system.GetContainer(ComponentName="Engineering Data")
neopreneRubber = engineeringData.ReadMaterial(
    Name="Neoprene Rubber",
    Source="Hyperelastic_Materials.xml")
neoHookeanPropertyData = neoHookeanPropertyData.GetPropertyData(Name="Neo-Hookean")
curveFit = neoHookeanPropertyData.CreateCurveFitting(
    Type="Neo-Hookean",
    Definition="")
uniaxialProperty = neopreneRubber.GetProperty(Name="Uniaxial Test Data")
curveFit.AddTestData(TestData=uniaxialProperty)
biaxialProperty = neopreneRubber.GetProperty(Name="Biaxial Test Data")
curveFit.AddTestData(TestData=biaxialProperty)
shearProperty = neopreneRubber.GetProperty(Name="Shear Test Data")
curveFit.AddTestData(TestData=shearProperty)
volumetricProperty = neopreneRubber.GetProperty(Name="Volumetric Test Data")
curveFit.AddTestData(TestData=volumetricProperty)
curveFit.RemoveTestData(TestData=uniaxialProperty)
curveFit.AddTestData(TestData=uniaxialProperty)
curveFit.Solve()
curveFit.CopyCoefficients(Destination=neoHookeanPropertyData)
curveFit.Delete()
```

CreateVariable

Include an additional variable in the property data for a material property.

Return  The new variable data that was created.

Type  DataReference

Required Arguments

Name  The new variable data name.

Type  string
Optional Arguments

Qualifiers
The optional dictionary of a qualifier name and it's corresponding value.

Type  Dictionary<string, string>

Delete
Deletes the material property data.

EndBatchUpdate
Marks the completion of a series of data modifications.

GetChartData
Returns a generated graph data for the specified source data.

Valid source data are:

    Material Property
    Material Property Data

Return  The graph data for the specified source data.

Type  DataReference

GetCurveFitting
Query to return the DataReference to the CurveFit used by some PropertyData.

Return  CurveFit DataReference

Type  DataReference

GetVariable
Returns a material variable of a given name from a material property data.

Return  The requested variable.

Type  DataReference

Required Arguments

Name  The name of the variable.

Type  string

Example
The following example creates new Engineering Data system and queries Coefficient of Thermal Expansion property data of the default material Structural Steel.

```csharp
ENGDTemplate = GetTemplate(TemplateName="EngData")
ENGDSystem = ENGDTemplate.CreateSystem(Position="Default")
```
ENGDContainer = ENGDSystem.GetContainer(ComponentName="Engineering Data")
StructSteel = ENGDContainer.GetMaterial(Name="Structural Steel")

# Get material
CTEProperty = StructSteel.GetProperty(Name="Coefficient of Thermal Expansion")
# Get material property data
CTEPropData = CTEProperty.GetPropertyData(
    Name="Coefficient of Thermal Expansion",
    Definition="Secant",
    Behavior="Isotropic")
# Get material variable
CTEVariable = CTEPropData.GetVariable(
    Name="Coefficient of Thermal Expansion")

**IsSuppressed**

Checks if an entity is suppressed.

Valid entities are:

- Material
- Material Property
- Material Property Data

**Return**  Returns true if the entity is suppressed, false otherwise.

**Type**  bool

**IsValid**

Validates a material property data and provides a message in case of invalid data.

**Return**  The flag that indicates if the material property data is valid.

**Type**  bool

**Required Arguments**

**Material**  The parent material of the material property data.

**Type**  DataReference

**Optional Arguments**

**Message**  The validation failure message.

**Type**  Output<string>

**SetSuppression**

Suppresses or Unsuppresses item.

Item can be suppressed to prevent it from being sent to a downstream cell in the system.

Following items can be suppressed:

- Material
- Material property
Material property data

**Required Arguments**

**Suppressed** The flag to specify if the item should be suppressed or unsuppressed.

  * Type `bool`

**MaterialVariable**

The entity to store material variable information.

**Properties**

**DatumColl**

The collection of the material variable data.

  * Type `DataReferenceSet`
  * Read Only `No`

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

  * Type `string`
  * Read Only `No`

**IsDependent**

The flag that indicates if the material variable is dependent, e.g., Density is temperature dependent.

  * Type `bool`
  * Read Only `No`

**LowerBoundUnit**

The unit of the lower bound data value in the material variable data.

  * Type `string`
  * Read Only `No`

**LowerBoundValue**

The lower bound data value in the material variable data.

  * Type `double`
  * Read Only `No`
**MaxValue**
The maximum value limit of the material variable data.

**Type** double

**Read Only** No

**MinValue**
The minimum value limit of the material variable data.

**Type** double

**Read Only** No

**Offset**
The offset value of the material variable data.

**Type** Quantity

**Read Only** No

**Scale**
The scale factor of the material variable data.

**Type** double

**Read Only** No

**TypeName**
The name of the material variable.

**Type** string

**Read Only** No

**UniqueData**
The collection of unique data values in the material variable data.

**Type** DataReferenceSet

**Read Only** No

**UpperBoundUnit**
The unit of the upper bound data value in the material variable data.

**Type** string

**Read Only** No
**UpperBoundValue**

The upper bound data value in the material variable data.

**Type** double

**Read Only** No

**Methods**

**Delete**

Deletes an additional variable in the property data from a material property.

**IsValid**

Validates a material variable data and provides a message in case of invalid data.

**Return** The flag that indicates if the material variable is valid.

**Type** bool

**Required Arguments**

**Material** The parent material of the variable.

**Type** DataReference

**MaterialPropertyData** The parent material property data of the variable.

**Type** DataReference

**Optional Arguments**

**Message** The validation failure message.

**Type** Output<string>

**Example**

```plaintext
template = GetTemplate(TemplateName="EngData") system = template.CreateSystem(Position="Default") container = system.GetContainer(ComponentName="Engineering Data") structSteel = container.GetMaterial(Name="Structural Steel") CTEProp = structSteel.GetProperty(Name="Coefficient of Thermal Expansion") CTEPropData = CTEProp.GetPropertyData( Name="Coefficient of Thermal Expansion", Definition="Secant", Behavior="Isotropic") CTEVariable = CTEPropData.GetVariable( Name="Coefficient of Thermal Expansion") valid = EngData.ValidateMaterialVariable(MaterialVariable=CTEVariable, Material=structSteel, MaterialPropertyData=CTEPropData)
```
External Connection

This container holds data for an instance of the External Process Connector.

Methods

**ExecuteOperation**

Command that wraps the script invoked by a custom GUI Operation

**Required Arguments**

- **Operation**
  - The name of the operation to execute
  - **Type** string

**GetExternalConnectionProperties**

A Query to return a reference to the ExternalConnectionProperties entity

- **Return**
  - A reference to the requested data entity.
  - **Type** DataReference

**ReadConfiguration**

Reads an External Connection configuration file.

**Required Arguments**

- **FilePath**
  - The full path to the configuration file.
  - **Type** string

**ReadParameterValues**

Populates all the values of the parameters of a given type within a container.

**Required Arguments**

- **ParameterType**
  - Type of parameter. Should be "input" or "output".
  - **Type** string
Data Entities

ExternalConnectionProperties

This entity holds the properties used to connect a Workbench project to an external process or application.

Properties

ConfigFile

The path to the configuration file defining application settings.

Type  string
Read Only  No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type  string
Read Only  No

WorkingDirectory

The path to the working directory. This is optional. If this value is not set, it defaults to the add-in directory corresponding to the system ("project_files/dpN/sysDir/addinDir")

Type  string
Read Only  No
External Data

This container holds data to expose external results or data files within the Workbench project.

Methods

**AddDataFile**

Adds a data file to the outline

*Return* Returns the reference to the FileData object

*Type*  DataReference

**Required Arguments**

*FilePath* The data file path

*Type*  string

**AddRepositoryFile**

Adds a data file to the outline

*Return* Returns the reference to the FileData object

*Type*  DataReference

**Required Arguments**

*File* The repository file

*Type*  DataReference

*Index* Inserts the data files in the specified location in the internal list. It is 0-based.

*Type*  int

**GetExternalLoadData**

Query to return the reference to the container's ExternalLoadData data entity.

*Return* A reference to the requested ExternalLoadData data entity.

*Type*  DataReference
**GetExternalLoadOutput**

Query to return the reference to the container's ExternalDataOutput data entity.

*Return*  
A reference to the requested ExternalLoadData data entity.

*Type*  
DataReference

**InsertDataFile**

Adds a data file to the outline.

*Return*  
Returns the reference to the FileData object.

*Type*  
DataReference

**Required Arguments**

*FilePath*  
The data file path.

*Type*  
string

*Index*  
Inserts the data files in the specified location in the internal list. It is 0-based.

*Type*  
int

**ModifyRepositoryFile**

Changes the file path of the existing FileData.

**Required Arguments**

*FileData*  
Reference to the FileData to be modified.

*Type*  
DataReference

*RepositoryFile*  
File to be modified in the FileData.

*Type*  
DataReference

**RereadDataFiles**

When you modify an External Data system's data file outside of the Workbench and you need to cause the Workbench to re-read the data file.

**ScanForFileChanges**

This command is useful, when you modify an External Data system's data file outside of the Workbench and you need to cause the Workbench to re-read the data file.

**Data Entities**

**ExternalLoadColumnData**

This entity contains information about the column data.
**Properties**

**DataType**

DataType of this column data

*Type*  
`string`

*Read Only*  
No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type*  
`string`

*Read Only*  
No

**Identifier**

Identifier of this column data. This will be used by the downstream applications

*Type*  
`string`

*Read Only*  
No

**Unit**

Unit of this column data

*Type*  
`string`

*Read Only*  
No

**ExternalLoadData**

This is the root level entity for the external data add-in

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type*  
`string`

*Read Only*  
No

**FilesData**

Contains the List of FileData references

*Type*  
`List<DataReference>`
External Data

Read Only  No

Methods

**DeleteFileData**

Removes a data file from the outline

**Required Arguments**

**FileData**  Reference to the FileData

  **Type**  DataReference

**DuplicateFileData**

Removes a data file from the outline

**Required Arguments**

**FileData**  Reference to the FileData

  **Type**  DataReference

**GetExternalLoadFileData**

Query to return the reference to the container's ExternalLoadFileData data entity.

**Return**  A reference to the requested ExternalLoadFileData data entity.

  **Type**  DataReference

**Required Arguments**

**Name**  Name of the ExternalLoadFileData entity

  **Type**  string

**ExternalLoadFileData**

This entity contains information for the DataFile added to the outline

**Properties**

**Description**

Contains the description about the Data file

  **Type**  string

  **Read Only**  No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.
**Type**: string

**Read Only**: No

**File**

FileReference maintained by the project

**Type**: DataReference

**Read Only**: No

**FileDataProperty**

Contains the reference to the FileDataPoperty

**Type**: DataReference

**Read Only**: No

**Master**

Makes the Data file as a Master or not. Only one Data file can be a Master.

**Type**: bool

**Read Only**: No

**Methods**

**GetDataProperty**

Query to return the reference to the container's ExternalLoadFileDataProperty data entity.

**Return**: A reference to the requested ExternalLoadFileData data entity.

**Type**: DataReference

**ModifyFileData**

Changes the file path of the existing FileData

**Required Arguments**

**FilePath**: FilePath to be modified in the FileData

**Type**: string

**SetAsMaster**

Changes the Master property on the FileData

**Required Arguments**

**Master**: controls the whether to make it as Master or not
Type   bool

**SetDelimiterCharacter**

Changes the Delimited Character on the FileDataProperty

**Required Arguments**

**Delimiter**   New Delimiter for the FileDataProperty

  *Type*   string

**FileDataProperty**   FileDataProperty that needs to be modified

  *Type*   DataReference

**SetDelimiterType**

Changes the DelimiterType on the FileDataProperty

**Required Arguments**

**Delimiter**   New DelimiterType for the FileDataProperty

  *Type*   DelimiterType

**DelimiterString**   New Delimiter string for the FileDataProperty

  *Type*   string

**FileDataProperty**   FileDataProperty that needs to be modified

  *Type*   DataReference

**Example**

```c
solution1.SetDelimiterType(
    FileData=fileData,
    FileDataProperty=fileDataProperty,
    Delimiter=DelimiterType.Comma,
    DelimiterString="",
)
```

**SetFormatType**

Changes the FormatType on the FileDataProperty

**Required Arguments**

**FileDataProperty**   FileDataProperty that needs to be modified

  *Type*   DataReference

**Type**   New Format Type for the FileDataProperty

  *Type*   FormatType
SetStartImportAtLine

Changes the Start Import at Line on the FileDataProperty

Required Arguments

FileDataProperty FileDataProperty that needs to be modified
Type DataReference

LineNumber New LineNumber for the FileDataProperty
Type int

ExternalLoadFileDataProperty

Contains information displayed on the properties view

Properties

ColumnsData

Contains the list references to the Column Data

Type List<DataReference>
Read Only No

ConversionOption

Contains the option on how to convert the data

Type VariableConversionOption
Read Only No

CoordinateSystemType

Specifies whether to use the Cartesian or Cylindrical coordinate system. The default value is Cartesian

Type CoordinateSystemType
Read Only No

DelimiterCharacter

Contains the delimiter character. Based on this value, number of columns are calculated

Type string
Read Only No

DelimiterType

Contains either the predefined delimiter or the user defined delimiter type
**Type**  
DelimiterType

**Read Only**  
No

**Dimensions**

You can choose to either import data from 2D or 3D models. If the 2D option is selected, you will be able to import data only at the X and Y coordinates. The Z coordinate is not supported for the 2D option.

**Type**  
DimensionsType

**Read Only**  
No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

**FileIdentifier**

A string that can be used to identify the file in the downstream Mechanical application. The data identifiers, from the Table pane, are appended to this string so that you can pick the correct source data in the downstream Mechanical application.

**Type**  
string

**Read Only**  
No

**FormatString**

Contains the FormatString. Based on this value, number of columns are calculated.

**Type**  
string

**Read Only**  
No

**FormatType**

This can be either Delimited or User-Defined. If the value is Delimited - Delimiter Character field is valid  
If the value is User-Defined - FormatString field is valid

**Type**  
FormatType

**Read Only**  
No

**LengthUnit**

The units of measurement to be used.

**Type**  
string

**Read Only**  
No

---

Release 15.0 - © SAS IP Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
**OriginX**

Contains the OriginX value

**Type** double

**Read Only** No

**OriginXUnit**

Contains the unit of the OriginX

**Type** string

**Read Only** No

**OriginY**

Contains the OriginY value

**Type** double

**Read Only** No

**OriginYUnit**

Contains the unit of the OriginY

**Type** string

**Read Only** No

**OriginZ**

Contains the OriginZ value

**Type** double

**Read Only** No

**OriginZUnit**

Contains the unit of the OriginZ

**Type** string

**Read Only** No

**ScaleX**

Contains the Scale X string

**Type** string

**Read Only** No
**ScaleXToolTip**

Contains the tool tip of the Scale X

**Type**  
string

**Read Only**  
No

**ScaleXValid**

Contains the whether the Scale X string is valid or not

**Type**  
bool

**Read Only**  
No

**ScaleY**

Contains the Scale Y string

**Type**  
string

**Read Only**  
No

**ScaleYToolTip**

Contains the tool tip of the Scale Y

**Type**  
string

**Read Only**  
No

**ScaleYValid**

Contains the whether the Scale Y string is valid or not

**Type**  
bool

**Read Only**  
No

**ScaleZ**

Contains the Scale Z string

**Type**  
string

**Read Only**  
No

**ScaleZToolTip**

Contains the tool tip of the Scale Z

**Type**  
string

**Read Only**  
No
**ScaleZValid**

Contains the whether the Scale Z string is valid or not

**Type**  
bool

**Read Only**  
No

**StartImportAtLine**

The line number at which you want the data import to start. Line numbers start at 1.

**Type**  
int

**Read Only**  
No

**ThetaXY**

Contains the ThetaXY value

**Type**  
double

**Read Only**  
No

**ThetaXYUnit**

Contains the unit of the ThetaXY

**Type**  
string

**Read Only**  
No

**ThetaYZ**

Contains the ThetaYZ value

**Type**  
double

**Read Only**  
No

**ThetaYZUnit**

Contains the unit of the ThetaYZ

**Type**  
string

**Read Only**  
No

**ThetaZX**

Contains the ThetaZX value

**Type**  
double

**Read Only**  
No
**ThetaZXUnit**

Contains the unit of the ThetaZX

**Type** string  
**Read Only** No

**Methods**

### GetColumnData

Query to return the reference to the container's ExternalLoaColumnData data entity.

**Return** A reference to the requested ExternalLoadFileData data entity.  
**Type** DataReference

**Required Arguments**

**Name** Name of the ExternalLoadColumnData entity  
**Type** string

### SetColumnDataType

Changes the column data type of the given ColumnData

**Required Arguments**

**ColumnData** ColumnData to be modified  
**Type** DataReference  
**DataType** New DataType for the ColumnData  
**Type** string

### SetCoordinateSystemType

Changes the CoordinateSystem type in the FileDataProperty

**Required Arguments**

**Type** New Coordinate System Type for the FileDataProperty  
**Type** CoordinateSystemType

### SetDimensionLengthUnit

Changes the Data Type on the specified X,Y,Z coordinate

**Required Arguments**

**Coordinate** Coordinate X,Y,Z
**SetDimensionType**
Changes the DimensionsType on the FileDataProperty

**Required Arguments**
- **Dimensions**  
  New DimensionsType for the FileDataProperty  
  **Type**  
  DimensionsType

**SetFormatString**
Changes the FormatString on the FileDataProperty

**Required Arguments**
- **Format**  
  New Format string for the FileDataProperty  
  **Type**  
  string

**SetLengthUnit**
Changes the Length Unit on the FileDataProperty

**Required Arguments**
- **Unit**  
  New Length Unit for the FileDataProperty  
  **Type**  
  string
External Model Setup

This container holds data to expose external model setup data within the Workbench project.

Methods

**AddDataFile**

Adds a data file to the outline

*Return* Returns the reference to the FileData object

*Required Arguments*

- **FilePath** The data file path
  
  *Type* string

**AddRepositoryFile**

Adds a data file to the outline

*Return* Returns the reference to the FileData object

*Required Arguments*

- **File** The repository file
  
  *Type* DataReference

- **Index** Inserts the data files in the specified location in the internal list. It is 0-based.
  
  *Type* int

**GetExternalModelData**

Query to return the reference to the container's ExternalLoadData data entity.

*Return* A reference to the requested ExternalLoadData data entity.

*Type* DataReference
GetExternalModelOutput

Query to return the reference to the container’s ExternalDataOutput data entity.

Return

A reference to the requested ExternalLoadData data entity.

Type DataReference

InsertDataFile

Adds a data file to the outline

Return

Returns the reference to the FileData object

Type DataReference

Required Arguments

FilePath

The data file path

Type string

Index

Inserts the data files in the specified location in the internal list. It is 0-based.

Type int

ModifyRepositoryFile

Changes the file path of the existing FileData

Required Arguments

FileData

Reference to the FileData to be modified

Type DataReference

RepositoryFile

File to be modified in the FileData

Type DataReference

RreadDataFiles

When you modify an External Model system’s data file outside of the Workbench and you need to cause the Workbench to re-read the data file.

ScanForFileChanges

This command is useful, when you modify an External Data system’s data file outside of the Workbench and you need to cause the Workbench to re-read the data file.

Data Entities

ExternalModelData

This is the root level entity for the external data add-in
**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**FilesData**

Contains the List of FileData references

- **Type**: List<DataReference>
- **Read Only**: No

**Methods**

**DeleteFileData**

Removes a data file from the outline

**Required Arguments**

- **FileData**: Reference to the FileData
  - **Type**: DataReference

**DuplicateFileData**

Removes a data file from the outline

**Required Arguments**

- **FileData**: Reference to the FileData
  - **Type**: DataReference

**GetExternalModelFileData**

Query to return the reference to the container's ExternalLoadFileData data entity.

**Return**

A reference to the requested ExternalLoadFileData data entity.

- **Type**: DataReference

**Required Arguments**

- **Name**: Name of the ExternalModelFileData entity
  - **Type**: string
**ExternalModelFileData**

This entity contains information for the DataFile added to the outline

**Properties**

**Description**

Contains the description about the Data file

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**File**

FileReference maintained by the project

<table>
<thead>
<tr>
<th>Type</th>
<th>DataReference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**FileDataProperty**

Contains the reference to the FileDataProperty

<table>
<thead>
<tr>
<th>Type</th>
<th>DataReference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**Methods**

**GetDataProperty**

Query to return the reference to the container's ExternalLoadFileDataProperty data entity.

<table>
<thead>
<tr>
<th>Return</th>
<th>A reference to the requested ExternalLoadFileData data entity.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>DataReference</td>
</tr>
</tbody>
</table>

**ModifyFileData**

Changes the file path of the existing FileData

**Required Arguments**

**FilePath**  
FilePath to be modified in the FileData
Type  string

**ExternalModelFileDataProperty**

Contains information displayed on the properties view

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type  string
Read Only  No

**LengthUnit**

The units of measurement to be used.

Type  string
Read Only  No

**NumberOfCopies**

Number of copies to be created by downstream consumer

Type  int
Read Only  No

**OriginX**

Contains the OriginX value

Type  double
Read Only  No

**OriginXUnit**

Contains the unit of the OriginX

Type  string
Read Only  No

**OriginY**

Contains the OriginY value

Type  double
Read Only  No
**OriginYUnit**
Contains the unit of the OriginY

*Type*  
string

*Read Only*  
No

**OriginZ**
Contains the OriginZ value

*Type*  
double

*Read Only*  
No

**OriginZUnit**
Contains the unit of the OriginZ

*Type*  
string

*Read Only*  
No

**ThetaXY**
Contains the ThetaXY value

*Type*  
double

*Read Only*  
No

**ThetaXYUnit**
Contains the unit of the ThetaXY

*Type*  
string

*Read Only*  
No

**ThetaYZ**
Contains the ThetaYZ value

*Type*  
double

*Read Only*  
No

**ThetaYZUnit**
Contains the unit of the ThetaYZ

*Type*  
string

*Read Only*  
No
**ThetaZX**
Contains the ThetaZX value

**Type**
`double`

**Read Only**
No

**ThetaZXUnit**
Contains the unit of the ThetaZX

**Type**
`string`

**Read Only**
No

**TransformOriginal**
Whether or not we are going to transform the first instance of the model

**Type**
`bool`

**Read Only**
No

**Methods**

**SetLengthUnit**
Changes the Length Unit on the FileDataProperty

**Required Arguments**

**Unit**
New Length Unit for the FileDataProperty

**Type**
`string`
Finite Element Modeler

This container holds data for an instance of the Finite Element Modeler.

Methods

AddMeshFile

This command adds a Mesh File to the input meshes collection of the specified Container. Note that we allow the User to import the same file multiple times. Finally, note that the default Unit System will be the one of the Assembly Mesh.

Return

The newly added Input Mesh.

Type  DataReference

Required Arguments

FilePath  The input mesh file path.

Type  string

ModelType  The format of the input mesh file.

Type  ModelType

Edit

This command will launch the FE Modeler Editor (if not already running) and will either import the list of input meshes or open the currently existing database. Note that the Editor can be run in either Interactive mode or Batch mode.

Optional Arguments

Interactive  An optional boolean (True by default) which indicates if the Editor must be launched in Interactive mode (with a GUI so that the User can interact with it). If False, the Editor is run in Batch mode.

Type  bool

Default Value  True

Exit

The Exit command will close the potentially opened Editor associated with the Container argument.
GetFEMInputMesh

Returns the reference of one of the container's InputMesh data entity. Note that you will retrieve one of its specialized classes: FEMUstreamInputMesh or FEMUserInputMesh.

Return
The reference of the InputMesh data entity

Type  DataReference

Required Arguments
Name  Name of the input mesh

Type  string

GetFEMMesh

Returns the reference of the container's FEMMesh data entity.

Return
The reference of the FEMMesh data entity

Type  DataReference

Required Arguments
Name  Name of the entity

Type  string

GetFEMModel

Returns the reference of the container's FEMModel data entity.

Return
The reference of the FEMModel data entity

Type  DataReference

Required Arguments
Name  Name of the entity

Type  string

GetFEMSetup

Returns the reference of the container's FEMSetup data entity.

Return
The reference of the FEMSetup data entity

Type  DataReference

Required Arguments
Name  Name of the entity

Type  string
**GetGeometry**

Returns the reference of the container's FEMGeometry data entity.

**Return**

The reference of the FEMGeometry data entity

**Type**  
DataReference

**Required Arguments**

**Name**  
Name of the entity

**Type**  
string

**SendCommand**

The SendCommand command will execute the script contents, specified in Command, in the FE Modeler Editor associated with the specified Container. Note that if the associated FE Modeler Editor is not running when this command is issued, it will be launched and eventually closed at the end of the command. This command requires the Editor (if already up and running) not to be busy.

**Required Arguments**

**Command**  
Command argument containing the command.

**Type**  
string

**Data Entities**

**FEMInputMesh**

This object represents an input mesh for the FE Model. This input mesh can either be a Mesh File added by the User or data coming from an upstream container through a connection. In the first case, the object will be of type 'UserInputMeshObject'; in the other case, the object will be of type 'UpstreamInputMeshObject'. Many useful properties can be retrieved from this input mesh object.

**Properties**

**BodyGrouping**

The body grouping property of the input mesh (when importing it from a file). This entity is displayed as a Property for the User to modify.

**Type**  
BodyGrouping

**Read Only**  
No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No
**FileFormat**

The format of the mesh file. Even if the input type is an upstream connection, there is an associated transfer file.

*Type* ModelType

*Read Only* Yes

**FileReference**

The file reference of the mesh file. Even if the input type is an upstream connection, there is an associated transfer file.

*Type* DataReference

*Read Only* Yes

**IDHandling**

The ID Handling property of the input mesh. This entity is displayed as a Property for the User to modify.

*Type* IDHandling

*Read Only* No

**UnitSystem**

The Unit System of the input mesh (when importing it from a file). This entity is displayed as a Property for the User to modify.

*Type* String[]

*Read Only* No

**Methods**

**Delete**

This method removes an existing input mesh from an FE Modeler Model component.

**SwitchInputOrder**

This method switches the displayed order of two input meshes in the collection of an FE Modeler Model component.

**Required Arguments**

*OtherInputFile* The reference of the second input mesh to switch

*Type* DataReference
**FEMMesh**

This output object is used for transferring the mesh associated with the generated Faceted representation of the Geometry. The mesh data will be stored in an ACMO-formatted file whose reference can be retrieved using the 'ACMOFile' public property. Note that this mesh can be morphed if a Parametric study has been performed in the FE Modeler Editor.

**Properties**

**ACMOFile**

The reference of the ACMO file representing the Associated Mesh of the Faceted Geometry.

- **Type**: DataReference
- **Read Only**: Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**FEMModel**

This object represents the Assembly Mesh as well as the main storage location for some internal data for the container.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**GeometryImportAnalysisType**

Analysis Type preference. Import 3D objects or 2D objects (objects must be in the x-y plane)

- **Type**: GeometryAnalysisType
- **Read Only**: No

**InputMeshes**

The list of all input meshes for the current model.

- **Type**: List<DataReference>
**Read Only** Yes

**UnitSystem**

The Unit System name of the Assembly Mesh. This entity is displayed as a Property for the User to modify.

*Type* String[]

**FEMSetup**

This output object is used for transferring various generated data: The Faceted representation of the Geometry will be stored in an 'fedb' file whose reference can be retrieved using the 'FEModelerFile' public property. The ANSYS Input file for the ANSYS solver will be stored in an 'inp' file whose reference can be retrieved using the 'ANSYSInputFile' public property. The NURBS representation of the geometry will be stored in an 'x_t' file whose reference can be retrieved using the 'ParasolidFile' public property. The Materials used in the mesh will be stored in an XML file whose reference can be retrieved using the 'FiniteElementModelMaterials' public property.

**Properties**

**ANSYSInputFile**

The reference of the 'inp' file representing the ANSYS input for the solver.

*Type* DataReference

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type* string

**FEModelerFile**

The reference of the 'fedb' file representing the Faceted Geometry.

*Type* DataReference

**FiniteElementModelMaterials**

The reference of the 'xml' file representing the Materials of the mesh.

*Type* DataReference
**ParasolidFile**

The reference of the ‘x_t’ file representing the NURBS Geometry.

- **Type**: DataReference
- **Read Only**: Yes

**FEMUpstreamInputMesh**

An input mesh object representing an upstream connection

**Properties**

**BodyGrouping**

The body grouping property of the input mesh (when importing it from a file). This entity is displayed as a Property for the User to modify.

- **Type**: BodyGrouping
- **Read Only**: No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**FileFormat**

The format of the mesh file. Even if the input type is an upstream connection, there is an associated transfer file.

- **Type**: ModelType
- **Read Only**: Yes

**FileReference**

The file reference of the mesh file. Even if the input type is an upstream connection, there is an associated transfer file.

- **Type**: DataReference
- **Read Only**: Yes

**IDHandling**

The ID Handling property of the input mesh. This entity is displayed as a Property for the User to modify.

- **Type**: IDHandling
Read Only No

**UnitSystem**

The Unit System of the input mesh (when importing it from a file). This entity is displayed as a Property for the User to modify.

Type String[

Read Only No

**FEMUserInputMesh**

An input mesh object representing a Mesh File added by the User.

Properties

**BodyGrouping**

The body grouping property of the input mesh (when importing it from a file). This entity is displayed as a Property for the User to modify.

Type BodyGrouping

Read Only No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type string

Read Only No

**FileFormat**

The format of the mesh file. Even if the input type is an upstream connection, there is an associated transfer file.

Type ModelType

Read Only Yes

**FileReference**

The file reference of the mesh file. Even if the input type is an upstream connection, there is an associated transfer file.

Type DataReference

Read Only Yes
**IDHandling**

The ID Handling property of the input mesh. This entity is displayed as a Property for the User to modify.

- **Type**: IDHandling
- **Read Only**: No

**UnitSystem**

The Unit System of the input mesh (when importing it from a file). This entity is displayed as a Property for the User to modify.

- **Type**: String[]
- **Read Only**: No

**Geometry**

This entity represents the transfer of the Faceted representation of the Geometry, as well as its associated Mesh, to a downstream system. This object is basically a wrapper for the MeshObject and SetupObject data entities.

**Properties**

**ACMOFile**

The reference of the ACMO file representing the associated Mesh of the Faceted Geometry. This file is ACMO-formatted.

- **Type**: DataReference
- **Read Only**: Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**FacetedGeometry**

The reference of the Geometry file representing the associated Mesh of the Faceted Geometry. This file is a .fedb format.

- **Type**: DataReference
- **Read Only**: Yes
FLUENT

FLUENT Setup
This container holds Setup data for an instance of FLUENT.

Methods

CopyLauncherSettings
Copies the FLUENT Launcher Settings from one FLUENT Setup or Solution container to another FLUENT Setup or Solution container.

Edit
Opens the FLUENT editor to allow modification of FLUENT data.

This command will open the editor only if one is not already open on this system. If this system's editor is already open and in interactive mode, then it will be raised to the front.

Exit
Exits the FLUENT editor.

If no editor is open on the component in question, this command will have no effect.

GetFluentLauncherSettings
Returns the Data Entity which contains the Setup container's settings for the FLUENT Launcher.

Import
Imports the FLUENT mesh and FLUENT case settings into the FLUENT editor from an existing FLUENT .msh or .cas file; replacing the current FLUENT mesh and FLUENT case settings. If the imported file contains only a mesh, then the FLUENT case settings will be set to FLUENT's default values.

ImportRepositoryFile
Imports the FLUENT mesh/case file from EKM Repository

Required Arguments

File Reference to Case/Mesh file for Fluent

Type DataReference

FileType Reference to Fluent's File type
**SendCommand**

Execute a scheme, FLUENT GUI, or FLUENT TUI command(s) in the currently open FLUENT session. FLUENT GUI commands cannot be run if the currently open FLUENT session is running in no GUI mode.

If the FLUENT editor is not open, it will be opened in no GUI mode before executing the command(s) and then closed after the command(s) is executed.

**Example**

The following code shows how to execute the commands in a FLUENT journal file:

```plaintext
>> setup1.SendCommand(Command="/file/read-journal \"E:\\WB Projects\\Fluent Case-Data Files\\elbow.jou\\" ")
```

**Data Entities**

**ChartVariable**

Entity representing a variable in Convergence Chart

**Properties**

**Color**

Color of the curve representing this variable entity.

- **Type**: Color
- **Read Only**: No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**LineWidth**

Line width of the curve representing this variable entity.

- **Type**: float
- **Read Only**: No

**QuantityName**

The variable quantity to display.

- **Type**: string
**Symbol Size**

Symbol size of the points on the curve.

**Type**

`uint`

**Read Only**

No

**Methods**

**DeleteChartVariable**

Deletes a specified chart variable.

**ConvergenceChart**

Entity to store a convergence chart information.

**Properties**

**AxisX**

Associated X Axis

**Type**

`DataReference`

**Read Only**

No

**AxisY**

Associated Y Axis

**Type**

`DataReference`

**Read Only**

No

**ChartType**

MonitorChartType: Residual or UserDefined

**Type**

`MonitorChartType`

**Read Only**

No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**

`string`

**Read Only**

No
**Variables**

Collection of variables to be plotted.

**Type**  
DataReferenceSet

**Read Only**  
No

**XAxis**

Label for the x-axis/Horizontal Axis.

**Type**  
string

**Read Only**  
No

**Methods**

**GetAxis**

Returns the axis for a specified convergence chart

**Return**  
The axis

**Type**  
DataReference

**Required Arguments**

**Name**  
Name of the Axis

**Type**  
string

**GetChartVariable**

Returns the chart variable of a given name from a convergence chart

**Return**  
The chart variable that matches the specified name.

**Type**  
DataReference

**Required Arguments**

**Name**  
The name of the chart variable.

**Type**  
string

**GetChartVariables**

Returns the collection of chart variables for a given convergence chart

**Return**  
A collection of the variables in the chart.

**Type**  
DataReferenceSet
FluentLauncherSettings

Allows you to specify FLUENT Launcher settings for Fluent Setup/Solution cells.

Properties

CachePassword

Specify whether or not you want to save the HP-MPI password for later use.

Type bool
Read Only No

ClusterHeadNode

Specify the name of the compute cluster head node.

This property is only available on Windows.

Type string
Read Only No

ClusterJobTemplate

A custom submission policy created by an IT administrator to define the job parameters for an application and employed by the cluster users to submit jobs. Relevant for Microsoft Job Scheduler.

Type string
Read Only No

ClusterNodeGroup

Used to set specific node group(s) on which to run the job. Relevant for Microsoft Job Scheduler.

Type string
Read Only No

ClusterProcessorUnit

Select the unit type (node/socket/core) on which the job would be running. Relevant for Microsoft Job Scheduler.

Type ProcessorUnit
Read Only No

ConvertUNCPath

Specify whether or not to convert a local path to a UNC path if any matching shared directory is found.

Type bool
**Read Only**  No

**CreateJobSubmissionXML**

Specify whether or not to create the job submission XML file.

This property is only available on Windows.

**Type**  bool

**Read Only**  No

**DisplayMesh**

Specify whether or not to show the mesh after the mesh file or the case/data file has been read into FLUENT.

**Type**  bool

**Read Only**  No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  string

**Read Only**  No

**EmbedGraphicsWindows**

Specify whether or not to embed the graphics windows in the FLUENT application window, or to have the graphics windows free-standing.

**Type**  bool

**Read Only**  No

**EnvPath**

Specify the list of environment variables to set before starting FLUENT.

**Type**  Dictionary<string, string>

**Read Only**  No

**Initialization**

Specify which initialization method to use.

**Type**  InitializationMethods

**Read Only**  No
**InitSolutionDataFile**

Specify the solution data file to be used for initializing the solution.

- **Type**: string
- **Read Only**: No

**Interconnect**

Specify the interconnect that you wish to use for parallel calculations (e.g., ethernet, myrinet, infniband, etc.).

- **Type**: string
- **Read Only**: No

**JobScheduler**

Use available Resource Manager (LSF, SGE, PBSPro) to launch FLUENT job

This property is only available when using compute nodes on Linux.

- **Type**: SchedulerSpecification
- **Read Only**: No

**JobSubmissionXmlFile**

Specify the name of the job submission XML file.

This property is only available on Windows.

- **Type**: string
- **Read Only**: No

**LSFCheckpointPeriod**

Specify the interval of automatic checkpointing for LSF.

This property is only available when using compute nodes on Linux.

- **Type**: int
- **Read Only**: No

**LSFQueue**

Specify the name of the LSF queue.

This property is only available when using compute nodes on Linux.

- **Type**: string
- **Read Only**: No
**LSFUseAutomaticCheckpointing**

Specify whether or not you want to use automatic checkpointing with LSF. The specific interval for checkpointing is determined by the LSFCheckpointPeriod property.

This property is only available when using compute nodes on Linux.

**Type**  
bool

**Read Only**  
No

**MachineFileName**

Specify the name of the file that contains a list of machine names to run the parallel job.

**Type**  
string

**Read Only**  
No

**MachineList**

Specify a list of machine names to run the parallel job.

**Type**  
string

**Read Only**  
No

**MachineSpec**

Specify a list of machine names, or a file that contains machine names.

**Type**  
MachineSpecification

**Read Only**  
No

**MpiType**

Specify the MPI type that you wish to use for the parallel calculations (e.g., MPICH2, HP, etc.).

**Type**  
string

**Read Only**  
No

**NumberOfProcessors**

Specify the number of processors you wish to use for the parallel calculations (e.g., 2, 4, etc.).

**Type**  
int

**Read Only**  
No

**NumberOfProcessorsMeshing**

Specify the number of processors you wish to use for the meshing parallel calculations (e.g., 2, 4, etc.).
**Type** \(\text{int}\)

**Read Only** No

**Precision**

Specify whether to use the single-precision or the double-precision solver.

**Type** \(\text{CasePrecision}\)

**Read Only** No

**PrePostOnly**

Specify whether or not you want to run FLUENT in PrePost mode, which only allows you to set up or postprocess a problem (i.e., no calculations can be performed)

**Type** \(\text{bool}\)

**Read Only** No

**RemoteFluentRootPath**

Specify the root path of the remote FLUENT Linux installation.

This property is only available on Windows.

**Type** \(\text{string}\)

**Read Only** No

**RemoteHostName**

Specify the name of the head machine on the remote Linux cluster.

This property is only available on Windows.

**Type** \(\text{string}\)

**Read Only** No

**RemoteRshCommand**

Specify the command to connect to the remote node (the default is RSH).

This property is only available on Windows.

**Type** \(\text{RshSpecification}\)

**Read Only** No

**RemoteRshOtherCommand**

Specify the custom RSH spawn command used to connect to the remote Linux machine.

This property is only available on Windows.
RemoteRunOnLinux

Specify whether or not you want to run your FLUENT simulation on 64-bit Linux machines.

This property is only available on Windows.

RemoteUseHost

Specify whether or not to use the remote cluster head node that FLUENT will connect to for spawning (e.g., via rsh or ssh).

Use the RemoteHostName property to specify the name of the remote cluster head node.

This property is only available on Windows.

RemoteUseWorkingDirectory

Specify whether or not to use a working directory for remote Linux nodes.

Use the RemoteWorkingDirectory property to specify the name of the working directory.

This property is only available on Windows.

RemoteWorkingDirectory

Specify the name of the working directory for remote Linux nodes.

This property is only available on Windows.

RunParallel

Specify whether or not you want to run the parallel version of FLUENT.
**SetupCompilationEnvironment**

Specify whether or not you want to define settings for compiling user-defined functions (UDFs) with FLUENT.

This property is only available on Windows.

**Type** bool

**Read Only** No

**SGEPE**

Specify the SGE parallel environment where you want to submit your FLUENT jobs.

This property is only available when using compute nodes on Linux.

**Type** string

**Read Only** No

**SGEQMaster**

Specify the name of the Qmaster host.

This property is only available when using compute nodes on Linux.

**Type** string

**Read Only** No

**SGEQueue**

Specify the name of the queue where you want to submit your FLUENT jobs.

This property is only available when using compute nodes on Linux.

**Type** string

**Read Only** No

**SGESettings**

Specify SGE Settings to be used.

This property is only available when using compute nodes on Linux.

**Type** string

**Read Only** No

**SGEUseSettings**

Specify whether or not to use the specified SGE Settings.

Use the SGESettings property to specify the SGE Settings.
This property is only available when using compute nodes on Linux.

**Type** \( \text{bool} \)

**Read Only** No

**ShowLauncher**

Specify whether or not to show FLUENT Launcher when FLUENT starts.

**Type** \( \text{bool} \)

**Read Only** No

**StartWhenResourcesAvailable**

Specify whether or not to start the FLUENT job when resources are available.

This property is only available on Windows.

**Type** \( \text{bool} \)

**Read Only** No

**UDFPath**

Specify the path to the UDF compilation script (available when SetupCompilationEnvironment is TRUE).

This property is only available on Windows.

**Type** \( \text{string} \)

**Read Only** No

**UseJobScheduler**

Specify whether or not to use a job scheduler to run FLUENT jobs.

**Type** \( \text{bool} \)

**Read Only** No

**UseLSFCheckpoint**

Specify whether or not to use LSF checkpointing.

This property is only available when using compute nodes on Linux.

**Type** \( \text{bool} \)

**Read Only** No

**UseLSFQueue**

Specify whether or not to use the LSF queue.

This property is only available when using compute nodes on Linux.
**UseSharedMemory**

Specify whether or not to use shared memory on the local machine or to use distributed memory on a cluster.

Type: bool
Read Only: No

**UseUpstreamLauncherSettings**

Specify whether or not the current system’s Solution cell should use FLUENT Launcher’s property settings from the current system’s Setup cell.

Type: bool
Read Only: No

**WorkbenchColorScheme**

Specify whether or not to use the Workbench color scheme in the graphics window, or the classic black background color.

Type: bool
Read Only: No

**SetupData**

Data entity of Setup cell. Allows you to change attributes of Setup cell.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string
Read Only: No

**GenerateSetupOutput**

Allows you to generate case file for Setup cell if mesh file is input or mesh operations are defined.

Type: bool
Read Only: No
**FLUENT Solution**

This container holds Solution data for an instance of FLUENT.

**Methods**

**Continue**

Continue solving from current solution. This command should only be used when the Solution component is either Interrupted or Up-to-Date.

**CopyLauncherSettings**

Copies the FLUENT Launcher Settings from one FLUENT Setup or Solution container to another FLUENT Setup or Solution container.

**Edit**

Opens the FLUENT editor to allow modification of FLUENT data.

This command will open the editor only if one is not already open on this system. If this system's editor is already open and in interactive mode, then it will be raised to the front.

**Exit**

Exits the FLUENT editor.

If no editor is open on the component in question, this command will have no effect.

**GetComponentSettingsForRsmDpUpdate**

This query is used to obtain the ComponentSettingsForRsmDpUpdate object for Journaling and Scripting.

**GetConvergenceChart**

Returns the convergence chart of a given name in a container.

Return: The convergence chart that matches the specified name.

Type: DataReference

**Required Arguments**

Name: The name of the convergence chart.

Type: string

**GetConvergenceCharts**

Returns the collection of convergence charts in a container. If no convergence charts are in the container, the collection is empty.
**Return**
Collection of the convergence charts in the container.

**Type**  DataReferenceSet

**GetFluentLauncherSettings**
Returns the Data Entity which contains the Setup container’s settings for the FLUENT Launcher.

**GetFluentSolutionProperties**
Returns the Data Entity which manages settings and data for the FLUENT Solution component.

**GetSolutionSettings**
This query is used to obtain the solution settings object for Journaling and Scripting

**ImportInitialData**
Imports an existing FLUENT .dat file as initial conditions for the FLUENT editor.
Discards the currently available Solution Data (and all stored previous solution points).

**ImportRepositoryFinalData**
Command to import final data file from repository

**Required Arguments**

**File**  Reference to Fluent data file

**Type**  DataReference

**MarkUpToDate**
Accepts an interrupted Solution as Up-to-Date.
The specified Solution component should be in Interrupted state. As a result of this command, the Solution will be marked Up-to-date.

**SendCommand**
Execute a scheme, FLUENT GUI, or FLUENT TUI command(s) in the currently open FLUENT session.
FLUENT GUI commands cannot be run if the currently open FLUENT session is running in no GUI mode.

If the FLUENT editor is not open, it will be opened in no GUI mode before executing the command(s) and then closed after the command(s) is executed.

**Example**
The following code shows how to execute the commands in a FLUENT journal file:

```plaintext
>> setup1.SendCommand(Command="/file/read-journal "E:\WB Projects\Fluent Case-Data Files\elbow.jou"")
```
Data Entities

AxisContinuous

Data entity for Scenegraph axis.

Properties

AutomaticRange

Boolean var for Automatic Range

**Type**

bool

**Read Only**

No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**

string

**Read Only**

No

QuantityName

Name of the quantity being plotted at this axis

**Type**

XAxisQuantity

**Read Only**

No

RangeMaximum

Max value of quantity at this axis

**Type**

double

**Read Only**

No

RangeMinimum

Min value of quantity at this axis

**Type**

double

**Read Only**

No

Scale

Scale of this axis

**Type**

Scale

**Read Only**

No
**Title**

Title of the axis

**Type** string

**Read Only** No

**ChartVariable**

Entity representing a variable in Convergence Chart

**Properties**

**Color**

Color of the curve representing this variable entity.

**Type** Color

**Read Only** No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

**Read Only** No

**LineWidth**

Line width of the curve representing this variable entity.

**Type** float

**Read Only** No

**QuantityName**

The variable quantity to display.

**Type** string

**Read Only** No

**SymbolSize**

Symbol size of the points on the curve.

**Type** uint

**Read Only** No
Methods

*DeleteChartVariable*

Deletes a specified chart variable.

*ChartVariableData*

Entity representing a variable in Convergence Chart

Properties

*DisplayText*

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

*VariableDimension*

Dimensions in string format

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

*VariableIndex*

A integer: variable index of given MonitorChartType

<table>
<thead>
<tr>
<th>Type</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

*VariableType*

MonitorChartType: Residual or UserDefined, an enum

<table>
<thead>
<tr>
<th>Type</th>
<th>MonitorChartType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

*ConvergenceChart*

Entity to store a convergence chart information.

Properties

*AxisX*

Associated X Axis
**Type** | DataReference  
---|---
**Read Only** | No  

**AxisY**
Associated Y Axis

**Type** | DataReference  
---|---
**Read Only** | No  

**ChartType**
MonitorChartType: Residual or UserDefined

**Type** | MonitorChartType  
---|---
**Read Only** | No  

**DisplayText**
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** | string  
---|---
**Read Only** | No  

**Variables**
Collection of variables to be plotted.

**Type** | DataReferenceSet  
---|---
**Read Only** | No  

**XAxis**
Label for the x-axis/Horizontal Axis.

**Type** | string  
---|---
**Read Only** | No  

**Methods**

**GetAxis**
Returns the axis for a specified convergence chart

**Return** | The axis
---|---
**Type** | DataReference
Required Arguments

**Name**  Name of the Axis

  **Type**  string

**GetChartVariable**

Returns the chart variable of a given name from a convergence chart

**Return**  The chart variable that matches the specified name.

  **Type**  DataReference

Required Arguments

**Name**  The name of the chart variable.

  **Type**  string

**GetChartVariables**

Returns the collection of chart variables for a given convergence chart

**Return**  A collection of the variables in the chart.

  **Type**  DataReferenceSet

**FluentLauncherSettings**

Allows you to specify FLUENT Launcher settings for Fluent Setup/Solution cells.

**Properties**

**CachePassword**

Specify whether or not you want to save the HP-MPI password for later use.

  **Type**  bool

  **Read Only**  No

**ClusterHeadNode**

Specify the name of the compute cluster head node.

This property is only available on Windows.

  **Type**  string

  **Read Only**  No

**ClusterJobTemplate**

A custom submission policy created by an IT administrator to define the job parameters for an application and employed by the cluster users to submit jobs. Relevant for Microsoft Job Scheduler.
**ClusterNodeGroup**

Used to set specific node group(s) on which to run the job. Relevant for Microsoft Job Scheduler.

- **Type**: string
- **Read Only**: No

**ClusterProcessorUnit**

Select the unit type (node/socket/core) on which the job would be running. Relevant for Microsoft Job Scheduler.

- **Type**: ProcessorUnit
- **Read Only**: No

**ConvertUNCPath**

Specify whether or not to convert a local path to a UNC path if any matching shared directory is found.

- **Type**: bool
- **Read Only**: No

**CreateJobSubmissionXML**

Specify whether or not to create the job submission XML file.

This property is only available on Windows.

- **Type**: bool
- **Read Only**: No

**DisplayMesh**

Specify whether or not to show the mesh after the mesh file or the case/data file has been read into FLUENT.

- **Type**: bool
- **Read Only**: No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No
**EmbedGraphicsWindows**

Specify whether or not to embed the graphics windows in the FLUENT application window, or to have the graphics windows free-standing.

Type: `bool`
Read Only: No

**EnvPath**

Specify the list of environment variables to set before starting FLUENT.

Type: `Dictionary<string, string>`
Read Only: No

**Initialization**

Specify which initialization method to use.

Type: `InitializationMethods`
Read Only: No

**InitSolutionDataFile**

Specify the solution data file to be used for initializing the solution.

Type: `string`
Read Only: No

**Interconnect**

Specify the interconnect that you wish to use for parallel calculations (e.g., ethernet, myrinet, infniband, etc.).

Type: `string`
Read Only: No

**JobScheduler**

Use available Resource Manager (LSF, SGE, PBSPro) to launch FLUENT job

This property is only available when using compute nodes on Linux.

Type: `SchedulerSpecification`
Read Only: No

**JobSubmissionXmlFile**

Specify the name of the job submission XML file.
This property is only available on Windows.

**Type** string

**Read Only** No

**LSFCheckpointPeriod**

Specify the interval of automatic checkpointing for LSF.

This property is only available when using compute nodes on Linux.

**Type** int

**Read Only** No

**LSFQueue**

Specify the name of the LSF queue.

This property is only available when using compute nodes on Linux.

**Type** string

**Read Only** No

**LSFUseAutomaticCheckpointing**

Specify whether or not you want to use automatic checkpointing with LSF. The specific interval for checkpointing is determined by the LSFCheckpointPeriod property.

This property is only available when using compute nodes on Linux.

**Type** bool

**Read Only** No

**MachineFileName**

Specify the name of the file that contains a list of machine names to run the parallel job.

**Type** string

**Read Only** No

**MachineList**

Specify a list of machine names to run the parallel job.

**Type** string

**Read Only** No

**MachineSpec**

Specify a list of machine names, or a file that contains machine names.
**MpiType**

Specify the MPI type that you wish to use for the parallel calculations (e.g., MPICH2, HP, etc.).

- **Type**: string
- **Read Only**: No

**NumberOfProcessors**

Specify the number of processors you wish to use for the parallel calculations (e.g., 2, 4, etc.).

- **Type**: int
- **Read Only**: No

**NumberOfProcessorsMeshing**

Specify the number of processors you wish to use for the meshing parallel calculations (e.g., 2, 4, etc.).

- **Type**: int
- **Read Only**: No

**Precision**

Specify whether to use the single-precision or the double-precision solver.

- **Type**: CasePrecision
- **Read Only**: No

**PrePostOnly**

Specify whether or not you want to run FLUENT in PrePost mode, which only allows you to set up or postprocess a problem (i.e., no calculations can be performed).

- **Type**: bool
- **Read Only**: No

**RemoteFluentRootPath**

Specify the root path of the remote FLUENT Linux installation.

This property is only available on Windows.

- **Type**: string
- **Read Only**: No
**RemoteHostName**

Specify the name of the head machine on the remote Linux cluster.

This property is only available on Windows.

**Type**     string

**Read Only**  No

**RemoteRshCommand**

Specify the command to connect to the remote node (the default is RSH).

This property is only available on Windows.

**Type**     RshSpecification

**Read Only**  No

**RemoteRshOtherCommand**

Specify the custom RSH spawn command used to connect to the remote Linux machine.

This property is only available on Windows.

**Type**     string

**Read Only**  No

**RemoteRunOnLinux**

Specify whether or not you want to run your FLUENT simulation on 64-bit Linux machines.

This property is only available on Windows.

**Type**     bool

**Read Only**  No

**RemoteUseHost**

Specify whether or not to use the remote cluster head node that FLUENT will connect to for spawning (e.g., via rsh or ssh).

Use the RemoteHostName property to specify the name of the remote cluster head node.

This property is only available on Windows.

**Type**     bool

**Read Only**  No

**RemoteUseWorkingDirectory**

Specify whether or not to use a working directory for remote Linux nodes.
Use the RemoteWorkingDirectory property to specify the name of the working directory.

This property is only available on Windows.

**Type**  
`bool`

**Read Only**  
No

**RemoteWorkingDirectory**

Specify the name of the working directory for remote Linux nodes.

This property is only available on Windows.

**Type**  
`string`

**Read Only**  
No

**RunParallel**

Specify whether or not you want to run the parallel version of FLUENT.

**Type**  
`bool`

**Read Only**  
No

**SetupCompilationEnvironment**

Specify whether or not you want to define settings for compiling user-defined functions (UDFs) with FLUENT.

This property is only available on Windows.

**Type**  
`bool`

**Read Only**  
No

**SGEPE**

Specify the SGE parallel environment where you want to submit your FLUENT jobs.

This property is only available when using compute nodes on Linux.

**Type**  
`string`

**Read Only**  
No

**SGEQMaster**

Specify the name of the Qmaster host.

This property is only available when using compute nodes on Linux.

**Type**  
`string`

**Read Only**  
No
SGEQueue
Specify the name of the queue where you want to submit your FLUENT jobs.
This property is only available when using compute nodes on Linux.
Type: string
Read Only: No

SGESettings
Specify SGE Settings to be used
This property is only available when using compute nodes on Linux.
Type: string
Read Only: No

SGEUseSettings
Specify whether or not to use the specified SGE Settings.
Use the SGESettings property to specify the SGE Settings.
This property is only available when using compute nodes on Linux.
Type: bool
Read Only: No

ShowLauncher
Specify whether or not to show FLUENT Launcher when FLUENT starts.
Type: bool
Read Only: No

StartWhenResourcesAvailable
Specify whether or not to start the FLUENT job when resources are available.
This property is only available on Windows.
Type: bool
Read Only: No

UDFPath
Specify the path to the UDF compilation script (available when SetupCompilationEnvironment is TRUE).
This property is only available on Windows.
Type: string
Read Only No

_useJobScheduler_

Specify whether or not to use a job scheduler to run FLUENT jobs.

Type bool
Read Only No

UseLSFCheckpoint

Specify whether or not to use LSF checkpointing.

This property is only available when using compute nodes on Linux.

Type bool
Read Only No

UseLSFQueue

Specify whether or not to use the LSF queue.

This property is only available when using compute nodes on Linux.

Type bool
Read Only No

UseSharedMemory

Specify whether or not to use shared memory on the local machine or to use distributed memory on a cluster.

Type bool
Read Only No

UseUpstreamLauncherSettings

Specify whether or not the current system's Solution cell should use FLUENT Launcher's property settings from the current system's Setup cell.

Type bool
Read Only No

WorkbenchColorScheme

Specify whether or not to use the Workbench color scheme in the graphics window, or the classic black background color.

Type bool
Read Only No
FluentSolutionProperties

Entity that manages settings and data for the FLUENT Solution component.

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string
Read Only: No

EnableDataInterpolation

It allows to generate the interpolation file at the end of calculation. It can be used as initial data for next calculation even if inout mesh data has changed.

Type: bool
Read Only: No

EnableSolutionMonitoring

It allows to generate the solution monitoring data which can be viewed using commands at Solution cell.

Type: bool
Read Only: No
Geometry

This container holds imported or generated geometry from an instance of DesignModeler.

Methods

**Edit**

The Edit command starts DesignModeler session, if it is not already running. If DesignModeler session is already running, this command brings the DesignModeler window in focus.

If a CAD file is assigned to geometry component, the file is loaded in the DesignModeler.

Available options:

| Interactive                      | This optional boolean type argument indicates whether DesignModeler is to be started in Interactive mode or batch mode. Its default value is True |

Optional Arguments

**Interactive**

Whether to edit in batch mode

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>True</td>
</tr>
</tbody>
</table>

**IsSpaceClaimGeometry**

Whether to edit geometry in SpaceClaim. If the user wants to use SpaceClaim as the geometry editor, then IsSpaceClaimGeometry should be set to true. The default value of IsSpaceClaimGeometry is false. If the user edits the geometry using DesignModeler then "IsSpaceClaimGeometry" will be false.

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>False</td>
</tr>
</tbody>
</table>

**Exit**

Exit command shuts down the running session of DesignModeler. Before shutting down, DesignModeler generates the un-generated features and saves its database.

DesignModeler session can not be closed if it is busy in generating features or seeking user's input. In such situations, this command throws an ApplicationBusyException exception.
**Export**

The export command exports geometry data in DesignModeler to the location specified in its FilePath argument. The export file CAD format is deduced from the extension of FilePath.

Available options:

- **FilePath**: This argument points to the location where the exported file should be saved. The filename extension specifies the export file format.

The command throws following exception:

- **CommandFailedException**: export file format is not supported
- **Export file location doesn't exist**: Export file location doesn't exist
- **File name contains invalid characters**: File name contains invalid characters
- **ApplicationBusyException**: When DesignModeler is busy.

**Required Arguments**

- **FilePath**: Output file path
  
  **Type**: string

**Example**

Suppose the geometry in the container "Geometry" needs to exported to IGES and STEP formats. This can be achieved by the following example.

```
geometry1 = GetDataContainer("Geometry")
geometry1.Export(FilePath="C:/Models/geometry1.iges")
geometry1.Export(FilePath=AbsUserPathName("Models/geometry1.step"))
```

**GetGeometryProperties**

Return a reference to DataEntity managing property settings of geometry container.

**Return**

Reference to DataEntity managing geometry properties

**Type**: DataReference

**Refresh**

Refresh command refreshes the input data in a geometry component by consuming all changed data from upstream (source) components. This command also updates the modified parameters in DesignModeler.

After successful execution of this command, the geometry component goes into "update required" state.

**Optional Arguments**

- **RepositoryFilesToRefresh**: Registered Repository files
  
  **Type**: List<DataReference>
**SendCommand**

The SendCommand sends javascript command string to DesignModeler for execution. If DesignModeler is not open, it will be launched to execute javascript commands, and then closed. If DesignModeler is already running, it will keep running after the command execution.

If DesignModeler is busy and not available for executing the javascript instruction, the SendCommand throws an ApplicationBusyException exception.

Available options:

- **Command** Javascript command string containing scripting commands for DesignModeler

The following command will add a sketch with an elliptical curve in DesignModeler

**Required Arguments**

<table>
<thead>
<tr>
<th>Command</th>
<th>Javascript command string</th>
</tr>
</thead>
</table>

**Type** string

**Example**

```javascript
system1 = GetSystem(Name="Geom")
geometry1 = system1.GetContainer(ComponentName="Geometry")
geometry1.SendCommand( Command = """
var ps1 = new Object();
ps1.Plane = agb.GetActivePlane();
ps1.XAxis = ps1.Plane.GetXAxis();
ps1.YAxis = ps1.Plane.GetYAxis();
ps1.Sk1 = ps1.Plane.NewSketch();
ps1.Sk1.Name = "Sketch1";
with (ps1.Sk1) { ps1.El7 = Ellipse( 8.0, 10.0, 9.0, 6.0, 5.0, 12.0); }
agb.Regen();""
```

**SetFile**

Adds a Geometry file to the Geometry System. The file processed by DesignModeler

Available options:

- **FilePath** Path of the Geometry File
- **PlugInName** PlugIn Name, in case of PlugIn mode geometry transfer

**Required Arguments**

<table>
<thead>
<tr>
<th>FilePath</th>
<th>Path of the Geometry File</th>
</tr>
</thead>
</table>

**Type** string

**Optional Arguments**

- **CreatedFileRef** Created File reference

**Type** Output<DataReference>
Geometry

PlugInName  PlugIn Name, in case of PlugIn mode Geometry transfer
  Type    string

RepositoryFileReference  Reference for the Geometry file if from repository
  Type    DataReference

Example

geometry1 = GetDataContainer("Geometry")
geometry1.SetFile("C:/Models/block.iges")
geometry2 = GetDataContainer("Geometry 1")
geometry2.SetFile(FilePath=AbsUserPathName("Models/block.prt.1"), PlugInName="ProEngineer[1]")

Stop

Stop command shuts down the running session of DesignModeler immediately, without saving its unsaved data.

The DesignModeler session can not be stopped if it is busy in importing or exporting CAD files. In such situations, this command throws an ApplicationException exception.

UpdateCAD

The UpdateCAD command updates geometry component using parameter values from DesignModeler. If the parameters are coming from external CAD systems through attach features in DesignModeler, then the attach feature is refreshed to update the parameters. The DesignModeler model is re-generated using the updated parameter values.

UpdateICManagerParam

Update IC Manager Param updates any of the exposed ICManager parameters

Required Arguments

QuantityName  Name of the Quantity To Update
  Type    string

QuantityValue  Double value of the Quantity to update
  Type    double

Data Entities

Geometry

Geometry data object
**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

`Type` string
`Read Only` No

**GeometryFilePath**

The location of file currently assigned to geometry component. DesignModeler process this file when started through Edit command.

`Type` string
`Read Only` No

**GeometryImportAnalysisType**

Analysis Type preference. Import 3D objects or 2D objects (objects must be in the x-y plane)

`Type` AnalysisType
`Read Only` No

**GeometryImportCadAssociativity**

Associativity preference. Indicates if action should be taken to allow associativity. This option is present because some CAD systems take too long to compute associativity.

`Type` bool
`Read Only` No

**GeometryImportCadAttributes**

Import Attributes preference. Allows import of CAD system attributes into the Mechanical application models. Enable this option to import Motion Loads.

`Type` bool
`Read Only` No

**GeometryImportCadAttributesFilter**

Import Attributes filter Key. (Displayed only when Attributes is selected.) This field can have any number of prefixes with each prefix delimited by a semicolon. If the filter is set to an empty string all applicable entities will be imported as Attributes.

`Type` string
`Read Only` No
**GeometryImportComparePartsOnUpdate**

Compare Parts On Update preference. Runs a post update comparison of parts from the original model and the new one. Marking those which have no topological or geometric changes as unmodified. This marking saves the time for remeshing.

**Type** ComparePartsOnUpdateMethod

**Read Only** No

**GeometryImportComparePartsTolerance**

Compare Parts Tolerance Preference. Sets one of three tolerance values for comparison when running Compare Parts On Update

**Type** ComparePartsTolerance

**Read Only** No

**GeometryImportCoordinateSystems**

Import Coordinate Systems preference. Specifies whether coordinate systems created in the CAD application should be imported into the Mechanical application.

**Type** bool

**Read Only** No

**GeometryImportDecomposeDisjointFaces**

Decompose Disjoint Face preference. Use to turn on/off the breaking of disjoint faces into multiple faces.

**Type** bool

**Read Only** No

**GeometryImportImportUsingInstances**

Import Using Instances preference. Processes a CAD model by honoring its part instances to produce faster attach times and smaller database sizes.

**Type** bool

**Read Only** No

**GeometryImportLineBodies**

Import Line Bodies preference. (If mixed dimension parts, Mixed Import Resolution preference is used.)

**Type** bool

**Read Only** No
GeometryImportMaterialProperties

Import Material Properties preference. Allows import of material data defined in the CAD system. Only a subset of material data will be imported. This will include Young’s Modulus, Poisson Ratio, Mass Density, Specific Heat, Thermal Conductivity and Thermal Expansion Coefficient. Limited additional data may be imported depending on CAD support.

Type: bool
Read Only: No

GeometryImportMixedResolutionOption

Mixed Import Resolution preference. Allows parts of mixed dimension to be imported as components of assemblies which have parts of different dimension.

Type: MixedImportPref
Read Only: No

GeometryImportNamedSelections

Import Named Selections preference. Creates a named selection based on data generated in the CAD system or in the DesignModeler application.

Type: bool
Read Only: No

GeometryImportNamedSelectionsFilter

Import Named Selections filter Key. (Displayed only when Named Selections is selected.) This field can have any number of prefixes with each prefix delimited by a semicolon. If the filter is set to an empty string all applicable entities will be imported as Named Selections.

Type: string
Read Only: No

GeometryImportParameters

Import Parameters preference. Allows user to turn on or off parameter processing.

Type: bool
Read Only: No

GeometryImportParametersFilter

Import Parameter filter Key. (Displayed only when Parameters is selected.) Allows user to specify a key that must appear at the beginning or end of a CAD parameter name to be imported. If the filter is set to an empty string all CAD parameters will be imported.

Type: string
Read Only: No
**GeometryImportProcessEnclosures**

Enclosure and Symmetry preference. Use to turn on/off the processing of enclosure and symmetry named selections.

**Type**  
bool

**Read Only**  
No

**GeometryImportSavePartFile**

Reader Mode Saves Updated File preference. When set to Yes, the interface will save the part file of a model at the end of an update process using the same file name in the same directory.

**Type**  
bool

**Read Only**  
No

**GeometryImportSmartUpdate**

Smart CAD Update preference. Speeds up refresh of models that have unmodified components. If set to Yes and changes are made to other preferences, these will not be respected if the component is smart updated.

**Type**  
bool

**Read Only**  
No

**GeometryImportSolidBodies**

Import Solid Bodies preference. (If mixed dimension parts, Mixed Import Resolution preference is used.)

**Type**  
bool

**Read Only**  
No

**GeometryImportSTLAngleAlgorithmCutAngle**

Cut Angle parameter for the Angle Algorithm

**Type**  
double

**Read Only**  
No

**GeometryImportSTLAngleAlgorithmToleranceAngle**

Tolerance Angle parameter for the Angle Algorithm

**Type**  
double

**Read Only**  
No

**GeometryImportSTLCurvatureAlgorithmIgnoreSecondaryNodes**

Ignore secondary nodes parameter for the Curvature Algorithm
**GeometryImportSTLCurvatureAlgorithmPlanesToleranceAngle**

Planes tolerance parameter for the Curvature Algorithm

*Type* double

*Read Only* No

**GeometryImportSTLCurvatureAlgorithmSharpEdgesAngle**

Sharp edges angle parameter for the Curvature Algorithm

*Type* double

*Read Only* No

**GeometryImportSTLSDTAlgorithm**

Analysis Type preference. Import 3D objects or 2D objects (objects must be in the x-y plane)

*Type* STLSDTAlgorithm

*Read Only* No

**GeometryImportSurfaceBodies**

Import Surface Bodies preference. (If mixed dimension parts, Mixed Import Resolution preference is used.)

*Type* bool

*Read Only* No

**GeometryImportWorkPoints**

Import Work Points preference. Specifies whether work points created in the CAD application should be imported into the Mechanical application.

*Type* bool

*Read Only* No

**PlugInName**

Current PlugInName - Returns TempPlugin if active DM session is editing. Otherwise return the real plugin name. This is also not persisted.

*Type* string

*Read Only* No
Geometry

**TeamcenterConnection**

The source string obtained from the Teamcenter, pointing to the NX geometry.

**Type**  
String

**Read Only**  
No
Graphics

This container holds charts and graphics objects in the project.

Data Entities

AxisContinuous

A chart axis that spans a set of continuous values. An example is an axis of an XY plot.

Properties

AutoScale

This property will define whether or not automatic scaling should be applied to the axis, or whether the RangeMin and RangeMax should be used.

Type: bool
Read Only: No

Label

The label of the axis.

Type: string
Read Only: No

Logarithmic

This property controls whether the axis scaling is to be logarithmic or linear. The default is linear scaling.

Type: bool
Read Only: No

RangeMax

The maximum range of the values in this axis.

Type: double
Read Only: No
**RangeMin**

The minimum range of the values in this axis.

**Type**  
**double**

**Read Only**  
**No**

**ShowGrid**

Defines whether or not to show a grid for this chart axis.

**Type**  
**bool**

**Read Only**  
**No**

**TitleBackgroundColor**

This defines the background color of an axis title. This is particularly useful when you want to be able to identify which variable is associated with which axis. The default is transparent.

**Type**  
**Color**

**Read Only**  
**No**

**Usability**

Determine whether this axis represents a usability axis. A usability axis presents discrete allowable values rather than continuous values.

**Type**  
**bool**

**Read Only**  
**No**

**AxisDiscrete**

A chart axis that represents a set of discrete values. An example is an axis of a bar chart.

**Properties**

**AutoScale**

This property will define whether or not automatic scaling should be applied to the axis, or whether the RangeMin and RangeMax should be used.

**Type**  
**bool**

**Read Only**  
**No**

**Label**

The label of the axis.

**Type**  
**string**

**Read Only**  
**No**
**RangeMax**

The index of the last division of the discrete data to be used. If this is -1 then it is undefined and will be determined dependent on the data.

**Type**  
uint

**Read Only**  
No

**RangeMin**

The index of the first division of the discrete data to be used. If this is -1 then it is undefined and will be determined dependent on the data.

**Type**  
uint

**Read Only**  
No

**ShowGrid**

Defines whether or not to show a grid for this chart axis.

**Type**  
bool

**Read Only**  
No

**TitleBackgroundColor**

This defines the background color of an axis title. This is particularly useful when you want to be able to identify which variable is associated with which axis. The default is transparent.

**Type**  
Color

**Read Only**  
No

**ChartXY**

This entity provides general properties for an XY (i.e. two dimensional) chart. Plotting details are determined from the associated variables and axes.

**Properties**

**Legend**

Reference to the Legend entity that is applied to the chart.

**Type**  
DataReference

**Read Only**  
No

**Style**

Reference to the RenderStyle entity that is applied to the chart.

**Type**  
DataReference
**Title**

The title of the chart.

**Type**  
string

**Read Only**  
No

**Variables**

The variables to be displayed in this chart. This can be a list of Variable, VariableXY or VariableXYZ data entities.

**Type**  
List<DataReference>

**Read Only**  
No

**XAxis**

The data reference to the x-axis.

**Type**  
DataReference

**Read Only**  
No

**XAxisSecondary**

The data reference to the secondary x-axis.

**Type**  
DataReference

**Read Only**  
No

**YAxis**

The data reference to the y-axis.

**Type**  
DataReference

**Read Only**  
No

**YAxisSecondary**

The data reference to the secondary y-axis.

**Type**  
DataReference

**Read Only**  
No

**ChartXYZ**

This chart is a canvas for an XYZ plot, the manner of plotting will be determined by the specified variables.
Properties

Legend
Reference to the Legend entity that is applied to the chart.

Type  DataReference
Read Only  No

Style
Reference to the RenderStyle entity that is applied to the chart.

Type  DataReference
Read Only  No

Title
The title of the chart.

Type  string
Read Only  No

Variables
The variables to be displayed in this chart. This can be a list of Variable, VariableXY or VariableXYZ data entities.

Type  List<DataReference>
Read Only  No

XAxis
The data reference to the X-Axis.

Type  DataReference
Read Only  No

YAxis
The data reference to the Y-Axis.

Type  DataReference
Read Only  No

ZAxis
The data reference to the Y-Axis.

Type  DataReference
CorrelationMatrix

A Correlation Matrix uses a tabular graphic to display the strength of the relationships between multiple parameters in a study.

Properties

CorrelationRange

This range defines the values and the distribution of correlation values to be applied to the color range. This defaults from -1 to 1.

Type List<float>
Read Only No

Legend

Reference to the Legend entity that is applied to the chart.

Type DataReference
Read Only No

Style

Reference to the RenderStyle entity that is applied to the chart.

Type DataReference
Read Only No

Title

The title of the chart.

Type string
Read Only No

Variables

The variables to be displayed in this chart. This can be a list of Variable, VariableXY or VariableXYZ data entities.

Type List<DataReference>
Read Only No

Legend

This entity provides a legend for chart data.
Properties

**BackgroundColor**

The background color of the legend.

Type: Color
Read Only: No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string
Read Only: No

**Enabled**

This property will enable or disable the legend.

Type: bool
Read Only: No

**ForegroundColor**

The foreground (border) color of the legend.

Type: Color
Read Only: No

**Orientation**

This property defines the orientation of the legend.

Type: OrientationStyle
Read Only: No

**MultiAxisChart**

Specialization of a chart to represent a parallel coordinate plot or a spider chart. Multi-axis charts use an independent axis for each supplied variable.

Properties

**ChartType**

Sets the type of rendering (e.g. Parallel Coordinate Plot, Spider Plot) for this multi-axis chart.

Type: ChartStyle
**Legend**
Reference to the Legend entity that is applied to the chart.

**Type**  
*DataReference*

**Read Only**  
No

**Style**
Reference to the RenderStyle entity that is applied to the chart.

**Type**  
*DataReference*

**Read Only**  
No

**Title**
The title of the chart.

**Type**  
*string*

**Read Only**  
No

**Variables**
The variables to be displayed in this chart. This can be a list of Variable, VariableXY or VariableXYZ data entities.

**Type**  
*List<DataReference>*

**Read Only**  
No

**PieChart**
Pie chart data object that allows us to represent a displayable pie chart.

**Properties**

**DivisionLabels**
In a multi-axis chart we are plotting each variable as an axis, but what we plot are actually displaying are the rows of each variable, as such we need labels for each row.

**Type**  
*DataReference*

**Read Only**  
No

**Legend**
Reference to the Legend entity that is applied to the chart.

**Type**  
*DataReference*
**Read Only**  No

**ShowPercentages**

Should the percentages for the slices be shown.

**Type**  bool

**Read Only**  No

**Style**

Reference to the RenderStyle entity that is applied to the chart.

**Type**  DataReference

**Read Only**  No

**Title**

The title of the chart.

**Type**  string

**Read Only**  No

**Variables**

The variables to be displayed in this chart. This can be a list of Variable, VariableXY or VariableXYZ data entities.

**Type**  List<DataReference>

**Read Only**  No

**RenderStyle**

This entity supplies the render properties for any graphics object.

**Properties**

**BarOffset**

This property controls the amount of space (relative to the BarWidth) before drawing a bar for this variable.

For example, if two variables are being drawn in a bar chart and you set the offset of the second variable to be 0.5, that variable will be shifted by half the BarWidth to avoid overlap.

**Type**  float

**Read Only**  No
**BarWidth**

This property controls the width of bars in a bar chart. The range of allowable values is 0 to 1, and sets the percentage of the available space used for the bars of the variable.

**Type**  
float

**Read Only**  
No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

**DotStyle**

The symbol to be used at each plot point.

**Type**  
DotStyles

**Read Only**  
No

**FillColors**

Define the fill color for this variable in the plot. All filled regions will use this color except where the style is defined as gradient in which case the GradientColor is used.

**Type**  
List<Color>

**Read Only**  
No

**FillStyle**

The shading to be used for any filled region.

**Type**  
FillStyles

**Read Only**  
No

**GradientAxis**

The axis that defines plot color if gradient shading is enabled. The axis must be continuous.

**Type**  
Axis

**Read Only**  
No

**LineColors**

Defines the line color of this variable in a plot. The first value in the list will be used if the line style is not gradient. Gradient line style will blend between the provided colors.
**LineColor**

The style of the line.

**Type**  
List\<Color>  

**LineStyle**

The style of the line.

**Type**  
LineStyles  

**LineWidth**

Sets the width of the line drawn for this variable in pixels.

**Type**  
uint  

**NumberOfColorBands**

Controls the number of color bands to be used in a gradient fill. A value of 0 (the default) will result in a continuous gradient.

**Type**  
uint  

**OutlineColors**

Define the outline colors for symbols. If not set (the default) then LineColors is used.

**Type**  
List\<Color>  

**ShowLinearInterpolationOfLines**

When set to true, causes the ends of a line chart to extend to the edge of the chart. This is primarily used to represent a constant line from a single value.

**Type**  
bool  

**Smoothing**

Enables smoothing of the rendered object. In 3D this results in a smoothed rather than faceted surface. In 2D this results in a smooth line rather than a straight line between points.

**Type**  
bool  

**Read Only**  
No
**SymbolSize**

Set the size of a symbol in pixels when a symbol is drawn for this variable. The rendered symbol size may be slightly smaller or larger than expected if symbol does not correctly fit into the specified number of pixels.

**Type**  
uint

**Read Only**  
No

**Variable**

The data entity that defines a variable to be plotted.

**Properties**

**AutoBounds**

Defines whether the bounds are to be used from BoundsMin/BoundsMax or whether they are to be automatically generated based on the data.

**Type**  
bool

**Read Only**  
No

**BoundsMax**

Defines the maximum rendered value for the data. Any larger values will be ignored.

**Type**  
float

**Read Only**  
No

**BoundsMin**

Defines the minimum rendered value for the data. Any smaller values will be ignored.

**Type**  
float

**Read Only**  
No

**DisplayAs**

Controls if this variable is displayed using lines, bars, etc.

**Type**  
VariableStyle

**Read Only**  
No

**FilterBoundsMax**

If bounds filtering is enabled, sets the maximum variable value that will cause it to be filtered from the plot. This is different from BoundsMin in that any variable that exceeds this value will be excluded from the plot. This primarily applies to Parallel Coordinate Plots.

**Type**  
float
**FilterBoundsMin**

If bounds filtering is enabled, sets the minimum variable value that will cause it to be filtered from the plot. This is different from BoundsMin in that any variable that exceeds this value will be excluded from the plot. This primarily applies to Parallel Coordinate Plots.

**Type** float

**IsFilterBoundsEnabled**

When this is true, any variables that are outside the filter bounds will be excluded from the plot. This primarily applies to Parallel Coordinate Plots.

**Type** bool

**IsIncludedInLegend**

Setting this parameter to 'false' will exclude this variable from any legend it may be included in. This property is only valid for XY and XYZ charts and will be ignored otherwise.

**Type** bool

**Label**

The label of the variable. This is optional and is typically determined from the name of the input variable.

**Type** string

**RelativeOrder**

Define the order of this variable among all the variables in a chart.

**Type** int

**VariableXY**

This is the base class for a data entity that defines a variable to be plotted.
Properties

AutoBounds
Defines whether the bounds are to be used from BoundsMin/BoundsMax or whether they are to be automatically generated based on the data.

Type: bool
Read Only: No

BoundsMax
Defines the maximum rendered value for the data. Any larger values will be ignored.

Type: float
Read Only: No

BoundsMin
Defines the minimum rendered value for the data. Any smaller values will be ignored.

Type: float
Read Only: No

DisplayAs
Controls if this variable is displayed using lines, bars, etc.

Type: VariableStyle
Read Only: No

FilterBoundsMax
If bounds filtering is enabled, sets the maximum variable value that will cause it to be filtered from the plot. This is different from BoundsMin in that any variable that exceeds this value will be excluded from the plot. This primarily applies to Parallel Coordinate Plots.

Type: float
Read Only: No

FilterBoundsMin
If bounds filtering is enabled, sets the minimum variable value that will cause it to be filtered from the plot. This is different from BoundsMin in that any variable that exceeds this value will be excluded from the plot. This primarily applies to Parallel Coordinate Plots.

Type: float
Read Only: No
**IsFilterBoundsEnabled**

When this is true, any variables that are outside the filter bounds will be excluded from the plot. This primarily applies to Parallel Coordinate Plots.

Type: bool
Read Only: No

**IsIncludedInLegend**

Setting this parameter to 'false' will exclude this variable from any legend it may be included in.

This property is only valid for XY and XYZ charts and will be ignored otherwise.

Type: bool
Read Only: No

**Label**

The label of the variable. This is optional and is typically determined from the name of the input variable.

Type: string
Read Only: No

**RelativeOrder**

Define the order of this variable among all the variables in a chart.

Type: int
Read Only: No

**VariableXYZ**

This is the base class for a data entity that defines a variable to be plotted.

**Properties**

**AutoBounds**

 Defines whether the bounds are to be used from BoundsMin/BoundsMax or whether they are to be automatically generated based on the data.

Type: bool
Read Only: No

**BoundsMax**

Defines the maximum rendered value for the data. Any larger values will be ignored.

Type: float
Read Only  No

**BoundsMin**

Defines the minimum rendered value for the data. Any smaller values will be ignored.

**Type**  float

**DisplayAs**

Controls if this variable is displayed using lines, bars, etc.

**Type**  VariableStyle

**FilterBoundsMax**

If bounds filtering is enabled, sets the maximum variable value that will cause it to be filtered from the plot. This is different from BoundsMin in that any variable that exceeds this value will be excluded from the plot. This primarily applies to Parallel Coordinate Plots.

**Type**  float

**FilterBoundsMin**

If bounds filtering is enabled, sets the minimum variable value that will cause it to be filtered from the plot. This is different from BoundsMin in that any variable that exceeds this value will be excluded from the plot. This primarily applies to Parallel Coordinate Plots.

**Type**  float

**IsFilterBoundsEnabled**

When this is true, any variables that are outside the filter bounds will be excluded from the plot. This primarily applies to Parallel Coordinate Plots.

**Type**  bool

**IsIncludedInLegend**

Setting this parameter to ‘false’ will exclude this variable from any legend it may be included in.

This property is only valid for XY and XYZ charts and will be ignored otherwise.

**Type**  bool

**Read Only**  No
**Label**

The label of the variable. This is optional and is typically determined from the name of the input variable.

**Type**  
string

**Read Only**  
No

**RelativeOrder**

Define the order of this variable among all the variables in a chart.

**Type**  
int

**Read Only**  
No
ICE

This container holds ICE data for an instance of IC Engine.

Methods

Refresh

Refresh ICE command.

Optional Arguments

UpstreamList  A list of upstream data containers that supply data to this cell
Type       List<DataContainerReference>

Reset

Reset ICE command.

Update

Update ICE command.

Data Entities

ICEData

ICE Data Object

Properties

CrankRadius

User input: Crank Radius
Type       Quantity
Read Only  No

CRLength

User input: Connecting Rod Length
Type       Quantity
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**DisplayText**

User input: Engine Speed

**EngineSpeed**

User input: Engine Speed

**EVO**

User input: Engine Speed

**ICCombustionSimulationType**

User input: IC Engine Combustion Simulation Type

**ICIVCandEVOOption**

User input: Option to get IVC and EVO values

**ICSimulationType**

User input: IC Engine Simulation Type

**IVC**

User input: Engine Speed
Read Only  No

**LiftCurvePath**
User input: Path to the valve lift curve
**Type**  string
**Read Only**  No

**MinLift**
User input: Minimum Lift
**Type**  Quantity
**Read Only**  No

**PistonOffset**
User input: Piston Offset
**Type**  Quantity
**Read Only**  No

---

**ICE Setup**
This container holds ICE setup data for an instance of IC Engine.

**Methods**

**Refresh**
Refresh ICE solver setup command.

**Optional Arguments**

**UpstreamList**  A list of upstream data containers that supply data to this cell

**Type**  List<DataContainerReference>

**Reset**
Reset ICE solver setup command.

**Update**
Update ICE solver setup command.
Data Entities

ICESetupData

ICE Setup Data Object

Properties

CrankAngleSelector

This is link for crank angle selection or KeyGrid dialog.

Type
string

Read Only
No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type
string

Read Only
No

ICKeyGridOption

Yes/No option for key grid

Type
YN

Read Only
No

PostIterationJournal

User input: Path to post iteration journal file

Type
string

Read Only
No

PreIterationJournal

User input: Path to Pre Iteration Journal File

Type
string

Read Only
No

SolverSettingEditor

This is link for solver setting dialog.

Type
string

Read Only
No
**UserBCProfileFile**

User input: Path to User Boundary Condition Profiles

**Type** string

**Read Only** No

**UserSettingsFilePath**

User input: Path to User Boundary Conditions and Monitor Path

**Type** string

**Read Only** No
ICEM

ICEM CFD
This container holds ICEM CFD data for an instance of ICEM.

Data Entities

SimulationGeneratedMesh
Mesh data object that resides in the ICEM CFD container.

Properties

DisplayText
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string
Read Only: No

Subsets
ICEM CFD creates Subsets instead of Parts from Named Selections if set.

Type: bool
Read Only: No

TransferFile
The ICEM CFD downstream files are *.msh "Imported FLUENT Mesh File Type", *.poly "POLYFLOWMesh", and *.inp "ANSYS Input File".

Type: DataReference
Read Only: No
IcePak

IcePak Setup

This container holds Setup data for an instance of IcePak.

Methods

Edit

User can launch the Icepak application by executing the Edit command

Optional Arguments

Interactive

Passing True will run Icepak in interactive mode

Type: bool

Default Value: True

SystemCoordinate

Optional parameter to specify the system coordinate to be displayed in the application title

Type: string

Exit

Close the Icepak session

Import

Import on the Setup container with an existing Icepak project or the compressed tgz Icepak project will launch an Icepak session and opens the imported project

Import on the Solution container allow users to set different set of case and data files for post processing. By default, the case and data files from the latest solution are available on the solution component.

Required Arguments

FilePath

Path to the project or tgz file when operated on Setup container. Path to the case file to be imported when operated on the Solution container

Type: string

Example

When the Import is called on the Setup container
Icepak.Import(FilePath=r"E:\DSModels\ICE\demo1_files\dp0\IPK\Icepak\IcepakProj")

When the Import is called on the Setup container with tzr file

Icepak.Import(FilePath=r"c:\Temp\test.tzr")

When the Import is called on the Solution container

Icepak.Import(FilePath=r"D:\IcepakProj00.cfd.cas")

**Data Entities**

**IcePakSetup**

Represents the Ic.epakSetup data entity

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**WorkbenchColorScheme**

Setting this property will force Ic.epak to use the Workbench color scheme. Unsetting this property will allow Ic.epak to use the default color scheme set inside Ic.epak.

- **Type**: bool
- **Read Only**: No

**IcePak Solution**

This container holds Solution data for an instance of Ic.epak.

**Methods**

**Import**

Import on the Setup container with an existing Ic.epak project or the compressed tzr Ic.epak project will launch an Ic.epak session and opens the imported project.
Import on the Solution container allow users to set different set of case and data files for post processing. By default, the case and data files from the latest solution are available on the solution component.

**Required Arguments**

**FilePath** Path to the project or tzr file when operated on Setup container. Path to the case file to be imported when operated on the Solution container

**Type** string

**Example**

When the Import is called on the Setup container

```python
Icepak.Import(FilePath=r"E:\DSModels\ICE\demol_files\dp0\IPK\Icepak\IcepakProj")
```

When the Import is called on the Setup container with tzr file

```python
Icepak.Import(FilePath=r"c:\Temp\test.tzr")
```

When the Import is called on the Solution container

```python
Icepak.Import(FilePath=r"D:\IcepakProj00.cfd.cas")
```
Mechanical APDL

This container holds data for a Mechanical APDL analysis.

Methods

**AddFile**

Copies the specified file to the working directory of the Mechanical APDL editor and registers the file with the Workbench project.

**AddInputFile**

Specifies an input file containing APDL commands for the Mechanical APDL editor for execution when the editor opens. Copies the file to the application working directory and registers it with the Workbench project.

**Edit**

Opens the Mechanical APDL editor to allow modification of Analysis data.

**Exit**

Exits the Mechanical APDL editor.

**GetAnalysisSettings**

Query to return the reference to the container's AnalysisSettings data entity.

**GetComponentSettingsForRsmDpUpdate**

This query is used to obtain the ComponentSettingsForRsmDpUpdate object for Journaling and Scripting

**GetMapdlInputFile**

Query to return the reference to the container's MapdlSetup data entity.

**GetMapdlSetup**

Query to return the reference to the container's MapdlSetup data entity.
GetSolutionSettings

This query is used to obtain the solution settings object for Journaling and Scripting.

SendCommand

Sends commands to the Mechanical APDL editor.

SwitchToBackgroundMode

Switch the Update in progress into background mode. This will enable operations that are not allowed during an Update in foreground mode (e.g. Project Save).

This command is not normally useful in a script. Journals may record the invocation of this command after an Update invoke, as the result of GUI activity while the Update is in progress. However, replay of these journals will always wait for the Update invoke to complete before invoking the next command, rendering this step ineffectual.

Data Entities

ExtendedComponentSettingsForRsmDpUpdate

Extended Component settings when solved as part of design point update via RSM Currently used for Addins with solvers that would like to take advantage of SMP.

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string
Read Only: No

LimitOnNumberOfCores

The number of processes that the RSM-based design point update should not exceed during execution.

Type: int
Read Only: No

SerialOnly

Indicates whether or not to execute only serially.

Type: bool
Read Only: No
**SharedMemoryParallel**

A boolean flag indicating whether or not to restrict the solver to using SMP, instead of distributed parallel.

- **Type**: `bool`
- **Read Only**: No

**UseLimitNumberOfCores**

Indicates whether to limit the number of processes for parallel execution.

- **Type**: `bool`
- **Read Only**: No

**MapdlInputFile**

Represents an input file for the Mechanical APDL editor.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: `string`
- **Read Only**: No

**Methods**

**Delete**

Deletes an input or reference file from the Mechanical APDL editor.

**PublishMapdlParameter**

Publishes a MAPDL variable located in an input file as a parameter.

**SwitchInputOrder**

Switches the order of two input files for the Mechanical APDL editor.

**UnpublishMapdlParameter**

a MAPDL variable located in an input file as a parameter.

**MapdlReferenceFile**

Represents a reference file for the Mechanical APDL editor.
Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type   string
Read Only   No

Methods

Delete

Deletes an input or reference file from the Mechanical APDL editor.

MapdlSetup

Represents the settings used to launch the Mechanical APDL editor.

Properties

CommandLineOptions

Additional command line options for the Mechanical APDL editor.

Type   string
Read Only   No

CustomExecutablePath

Calls a custom ANSYS executable.

Type   string
Read Only   No

DatabaseMemory

Defines the portion of workspace (memory) to be used for the database. The default is 512 MB for 64-bit machines, 256 MB for 32-bit machines.

Type   int
Read Only   No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type   string
Read Only   No

**Distributed**

Enables Distributed ANSYS.

**Type**   bool

**Read Only**   No

**DownloadDistributedFiles**

Indicates whether or not to download files from slave node scratches during distributed solves

**Type**   bool

**Read Only**   No

**GPUAccelerator**

Specifies the type of GPU Accelerators. By default this is set to none meaning no GPU acceleration is used.

**Type**   GPUAccelerator

**Read Only**   No

**Graphics**

Specifies the type of graphics device. This option applies only to interactive mode. For UNIX/Linux systems, graphics device choices are X11, X11C, or 3D. For Windows systems, graphics device options are WIN32 or WIN32C, or 3D.

**Type**   string

**Read Only**   No

**JobName**

Specifies the initial jobname, a name assigned to all files generated by the program for a specific model. If you omit the -j option, the jobname is assumed to be file.

**Type**   string

**Read Only**   No

**License**

Defines which ANSYS product will run during the session (ANSYS Multiphysics, ANSYS Structural, etc.).

**Type**   string

**Read Only**   No
**MachineList**

Specifies the machines on which to run a Distributed ANSYS analysis.

**Type** string

**Read Only** No

**MPIType**

Defines which type of MPI the solver should use.

**Type** MPIType

**Read Only** No

**NumberOfGpusPerMachine**

Specifies the number of accelerators to use when running with GPU Acceleration.

**Type** int

**Read Only** No

**Processors**

Specifies the number of processors to use when running Distributed ANSYS or Shared-memory ANSYS.

**Type** int

**Read Only** No

**ReadStartAns**

Specifies whether the program reads the start121.ans file at start-up.

**Type** bool

**Read Only** No

**WorkspaceMemory**

Specifies the total size of the workspace (memory) in megabytes. If you omit the -m option, the default is 1 GB (1024 MB) for 64-bit machines, 512 MB for 32-bit machines.

**Type** int

**Read Only** No
Mechanical Enhanced Model

This container holds Imported Section data for an instance of ANSYS Mechanical.

Methods

Edit

Opens the Mechanical editor and attaches geometry.

Optional Arguments

Interactive

Specify if Mechanical will open in interactive mode. The default value is true.

Type: bool

Default Value: True

Example

To edit the enhanced model component with default optional parameter values:

```python
enhancedModel.Edit()
```

Or by specifying optional parameter values:

```python
enhancedModel.Edit(Interactive=True)
```

Exit

Exit the Mechanical editor

Optional Arguments

SaveDatabase

Indicates whether the Mechanical database will be saved prior to exiting

Type: bool

Default Value: True
Mechanical Model

This container holds Model data for an instance of ANSYS Mechanical.

Methods

Edit

Opens the Mechanical editor and attaches geometry.

Optional Arguments

**Hidden**

Specify if Mechanical will open in hidden mode. The default value is false.

- **Type**: bool
- **Default Value**: False

**Interactive**

Specify if Mechanical will open in interactive mode. The default value is true.

- **Type**: bool
- **Default Value**: True

Example

To edit the model component with default optional parameter values:

```python
model.Edit()
```

Or by specifying optional parameter values:

```python
model.Edit(Interactive=True)
```

Exit

Exit the Mechanical editor

Optional Arguments

**SaveDatabase**

Indicates whether the Mechanical database will be saved prior to exiting

- **Type**: bool
- **Default Value**: True

Export

Exports a .dsdb file for the model component.

Required Arguments

**FilePath**

The path and name of the .dsdb file to be written.
**ExportASMJournal**

Exports an ASM Journal (.wbjn) and supporting files for the model component.

**Required Arguments**

- **FilePath**
  The path and name of the .wbjn file to be written. Supporting files written alongside.
  
  Type: string

**ExportGeometry**

Exports a PartManager database (.pmdb) file for the geometry in the model component.

**Required Arguments**

- **FilePath**
  The path and name of the .pmdb file to be written.
  
  Type: string

**ExportMesh**

Exports a .acmo file for the mesh in the model component.

**Required Arguments**

- **FilePath**
  The path and name of the .acmo file to be written.
  
  Type: string

**GetMechanicalMesh**

Query to return the reference to the container's MechanicalMesh data entity.

**Return**

A reference to the requested MechanicalMesh data entity.

Type: DataReference

**GetMechanicalMeshFile**

Query to return the reference to the container's MechanicalMeshFile data entity.

**Return**

A reference to the requested MechanicalMeshFile data entity.

Type: DataReference

**GetMechanicalModel**

Query to return the reference to the container's MechanicalModel data entity.

**Return**

A reference to the requested MechanicalModel data entity.

Type: DataReference
**GetMechanicalSystemType**

Query to return the reference to the container's MechanicalSystemType data entity.

**Return**

A reference to the requested MechanicalSystemType data entity.

**Type** DataReference

**GetSimulationImportOptions**

Query to return the data reference to the simulation import options

**Return**

The Data Entity containing settings for this component.

**Type** DataReference

**Required Arguments**

**Name**

The entity of interest.

**Type** string

**SendCommand**

Executes a JScript command in the Mechanical editor.

If the Mechanical editor is not open, then the editor's GUI will not be available, causing some commands to fail. In this case, consider calling the container's Edit method to open the editor before using SendCommand.

Furthermore, if the Mechanical editor is not open, SendCommand will start the editor without the GUI, issue the specified command, and then close the editor. For multiple SendCommands, this may degrade performance.

**Required Arguments**

**Command**

Command argument containing the command.

**Type** string

**Example**

To execute some arbitrary command (in this case, causing a dialog box to appear) in the Mechanical editor:

```javascript
model.SendCommand(Command="WBScript.Out("My Text",true);"
```

To run a JScript file already saved to disk using Run Macro from Mechanical:

```javascript
setup.SendCommand(Command="WB.AppletList.Applet("DSApplet").App.Script.doToolsRunMacro("C:\macro.js")"
```
Data Entities

**CDBImportSettingsEntity**

(Beta) Entity to control the Skin Detection algorithm for importing cdb files.

**Properties**

**CutAngle**

Only displayed if Forbid Close Components equals Yes. It is the angle used to cut closed surfaces to separate the elements into components.

Type: Quantity  
Read Only: No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string  
Read Only: No

**ForbidCloseComponents**

Option to split closed surfaces into several components. If Yes then the algorithm cuts closed surfaces into several components. It provides a simple method to avoid problematic faces on closed surfaces.

Type: bool  
Read Only: No

**GeometryImportAnalysisType**

The geometry attach type for the CDB file

Type: GeometryAttachType  
Read Only: No

**NodalComponentKey**

If the nodal components will be processed during CDB components the nodal component key will allow filtering of which components to process

Type: string  
Read Only: No

**ProcessNodalComponents**

Should the nodal components be processed during CDB conversion
**ToleranceAngle**

The tolerance angle to separate two elements into separate components. If the angle between the normals of two adjacent elements is less than or equal to the Tolerance Angle then the two elements are in the same component, otherwise, they are separated.

**GeneralModelAssemblyProperties**

Class used to expose general properties for the model assembly workflow

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**LengthUnit**

Length unit of assembled model

**MechanicalMesh**

This is the mesh data entity object that will exist in the model container.

**Properties**

**Caption**

Caption used to identify the mesh.

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.
Type          string
Read Only     No

MeshId

Mesh identifier that corresponds with Mesh ID in Mechanical
Type          int
Read Only     No

MechanicalModel

The model data entity in the model container.

Properties

AllowMeshing

Property used to prevent a re meshing in model assembly workflows
Type          bool
Read Only     No

Caption

Caption to identify the model
Type          string
Read Only     No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.
Type          string
Read Only     No

EdaFile

Reference to a material database
Type          DataReference
Read Only     No

File

Reference to the Mechanical database
Type          DataReference
**Read Only**  No

**IsResetNotRequired**

Property used to prevent cascading resets on downstream containers

Type  bool
Read Only  No

**ModelId**

Model identifier that corresponds with the ID on Model tree node in Mechanical

Type  int
Read Only  No

**PrototypId**

Prototype identifier which corresponds to the ID on the Geometry tree node in Mechanical

Type  int
Read Only  No

**MechanicalSystemType**

This entity provides string based information about the physics, analysis, and solver settings for the Mechanical system component.

**Properties**

**AnalysisTypeDisplayString**

The string which represents current analysis type setting.

Type  string
Read Only  Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type  string
Read Only  No

**PhysicsTypeDisplayString**

The string which represents current physics type setting.

Type  string
Read Only  Yes

**SolverTypeDisplayString**

The string which represents current solver type setting.

**Type**  string
**Read Only**  Yes

**ModelOutputSettingsForACP**

Entity to control the properties of mesh file generated by Mechanical Model.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  string
**Read Only**  No

**LengthUnit**

Length unit to write mesh file.

**Type**  string
**Read Only**  No

**SimulationImportOptions**

Import options for model assembly available on the downstream model.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  string
**Read Only**  No

**NumberOfCopies**

By default this value is 0, specifies how many additional copies of the upstream model are needed.

**Type**  int
**Read Only**  No
<table>
<thead>
<tr>
<th><strong>OriginX</strong></th>
<th>Displacement along the X axis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Quantity</td>
</tr>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>OriginY</strong></th>
<th>Displacement along the Y axis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Quantity</td>
</tr>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>OriginZ</strong></th>
<th>Displacement along the Z axis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Quantity</td>
</tr>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Source</strong></th>
<th>The upstream model source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>DataReference</td>
</tr>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>ThetaXY</strong></th>
<th>Rotation about the XY plane</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Quantity</td>
</tr>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>ThetaYZ</strong></th>
<th>Rotation about the YZ plane</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Quantity</td>
</tr>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>ThetaZX</strong></th>
<th>Rotation about the ZX plane</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Quantity</td>
</tr>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>
Mechanical Results

This container holds Results data for an instance of ANSYS Mechanical.

Methods

Edit

Opens the Mechanical editor and attaches geometry.

Optional Arguments

Interactive Specify if Mechanical will open in interactive mode. The default value is true.

Type bool

Default Value True

Example

To edit the results component with default optional parameter values.

    result.Edit()

Or by specifying optional parameter values:

    result.Edit(Interactive=True)

Exit

Exit the Mechanical editor

Optional Arguments

SaveDatabase Indicates whether the Mechanical database will be saved prior to exiting

Type bool

Default Value True

GetMechanicalSystemType

Query to return the reference to the container's MechanicalSystemType data entity.

Return A reference to the requested MechanicalSystemType data entity.

Type DataReference

GetSimulationResultFile

Query to return the component reference for a given container and component base name.

Return A reference to the requested data entity.
SendCommand

Executes a JScript command in the Mechanical editor.

If the Mechanical editor is not open, then the editor's GUI will not be available, causing some commands to fail. In this case, consider calling the container's Edit method to open the editor before using SendCommand.

Furthermore, if the Mechanical editor is not open, SendCommand will start the editor without the GUI, issue the specified command, and then close the editor. For multiple SendCommands, this may degrade performance.

Required Arguments

Command

Command argument containing the command.

Type string

Example

To execute some arbitrary command (in this case, causing a dialog box to appear) in the Mechanical editor:

```javascript
model.SendCommand(Command="WBScript.Out("My Text",true);"
```

To run a JScript file already saved to disk using Run Macro from Mechanical:

```javascript
setup.SendCommand(Command="WB.AppletList.Applet("DSApplet").App.Script.doToolsRunMacro("C:\\macro.js")")
```

Mechanical Setup

This container holds Set Up data for an instance of ANSYS Mechanical.

Methods

Edit

Opens the Mechanical editor and attaches geometry.

Optional Arguments

Interactive

Specify if Mechanical will open in interactive mode. The default value is true.

Type bool

Default Value True

Example

To edit the setup component with default optional parameter values.
setup.Edit()

Or by specifying optional parameter values:

setup.Edit(Interactive=True)

**Exit**

Exit the Mechanical editor

**Optional Arguments**

**SaveDatabase** Indicates whether the Mechanical database will be saved prior to exiting

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>True</td>
</tr>
</tbody>
</table>

**Export**

Writes either an APDL input file (for use with the ANSYS solver) or a CAE Representation file (for use with any solver).

**Required Arguments**

**Path** If the SetupDataType is 'InputFile', this should be a file path. If the SetupDataType is 'CAERepresentation', this should be a folder path.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
</table>

**SetupDataType** Type of setup data to write to disk. Available options are: 'InputFile' and 'CAERepresentation'.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
</table>

**GetMechanicalSetupFile**

Query to return the reference to the container's MechanicalSetupFile data entity.

**Return** A reference to the requested MechanicalSetupFile data entity.

<table>
<thead>
<tr>
<th>Type</th>
<th>DataReference</th>
</tr>
</thead>
</table>

**GetMechanicalSystemType**

Query to return the reference to the container's MechanicalSystemType data entity.

**Return** A reference to the requested MechanicalSystemType data entity.

<table>
<thead>
<tr>
<th>Type</th>
<th>DataReference</th>
</tr>
</thead>
</table>
**SendCommand**

Executes a JScript command in the Mechanical editor.

If the Mechanical editor is not open, then the editor's GUI will not be available, causing some commands to fail. In this case, consider calling the container's Edit method to open the editor before using SendCommand.

Furthermore, if the Mechanical editor is not open, SendCommand will start the editor without the GUI, issue the specified command, and then close the editor. For multiple SendCommands, this may degrade performance.

**Required Arguments**

- **Command**
  - Command argument containing the command.
  - **Type**: string

**Example**

To execute some arbitrary command (in this case, causing a dialog box to appear) in the Mechanical editor:

```javascript
model.SendCommand(Command="WBScript.Out("My Text",true);")
```

To run a JScript file already saved to disk using Run Macro from Mechanical:

```javascript
setup.SendCommand(Command="WB.AppletList.Applet("DSApplet").App.Script.doToolsRunMacro("C:\macro.js")")
```

**SetDesignAssessmentFile**

Set the design assessment

**Required Arguments**

- **FilePath**
  - The design assessment file's path.
  - **Type**: string

**Data Entities**

**DesignAssessmentSetupSettingsType**

The data entity that holds the setup properties for a Design Assessment.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
Read Only No

**SolverTarget**

The type of assessment to be performed.

Type **string**

Read Only No

**UserAttributeFile**

The file to specify attributes to perform a User Defined design assessment.

Type **string**

Read Only No

---

**Mechanical Solution**

This container holds Solution data for an instance of ANSYS Mechanical.

**Methods**

**Edit**

Opens the Mechanical editor and attaches geometry.

**Optional Arguments**

**Interactive** Specify if Mechanical will open in interactive mode. The default value is true.

Type **bool**

Default Value True

**Example**

To edit the solution component with default optional parameter values.

```python
solution.Edit()
```

Or by specifying optional parameter values:

```python
solution.Edit(Interactive=True)
```

**Exit**

Exit the Mechanical editor

**Optional Arguments**

**SaveDatabase** Indicates whether the Mechanical database will be saved prior to exiting
Type: bool
Default Value: True

**Export**

Exports the results(*.db, *.rth, *.rmg, and *.rst) files to given DirectoryPath.

**Required Arguments**

DirectoryPath: Directory path to export the result files to.
Type: string

**GetComponentSettingsForRsmDpUpdate**

This query is used to obtain the ComponentSettingsForRsmDpUpdate object for Journaling and Scripting

**GetExpertProperties**

This query is used to obtain the ExpertProperties object for Journaling and Scripting

**GetMechanicalSystemType**

Query to return the reference to the container’s MechanicalSystemType data entity.

Return: A reference to the requested MechanicalSystemType data entity.
Type: DataReference

**GetSolutionSettings**

Returns a DataReference to the Solution Settings object for this container

Return: The Data Entity containing settings for this component.
Type: DataReference

**SendCommand**

Executes a JScript command in the Mechanical editor.

If the Mechanical editor is not open, then the editor’s GUI will not be available, causing some commands to fail. In this case, consider calling the container’s Edit method to open the editor before using Send-Command.

Furthermore, if the Mechanical editor is not open, SendCommand will start the editor without the GUI, issue the specified command, and then close the editor. For multiple SendCommands, this may degrade performance.

**Required Arguments**

Command: Command argument containing the command.
**Type**  
string

**Example**
To execute some arbitrary command (in this case, causing a dialog box to appear) in the Mechanical editor:

```python
model.SendCommand(Command="WBScript.Out("My Text",true);" )
```

To run a JScript file already saved to disk using Run Macro from Mechanical:

```python
setup.SendCommand(Command="WB.AppletList.Applet("DSApplet").App.Script.doToolsRunMacro("C:\\macro.js")")
```

**Data Entities**

**ExtendedComponentSettingsForRsmDpUpdate**

Extended Component settings when solved as part of design point update via RSM Currently used for Addins with solvers that would like to take advantage of SMP

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

**LimitOnNumberOfCores**

The number of processes that the RSM-based design point update should not exceed during execution.

**Type**  
int

**Read Only**  
No

**SerialOnly**

Indicates whether or not to execute only serially.

**Type**  
bool

**Read Only**  
No

**SharedMemoryParallel**

A boolean flag indicating whether or not to restrict the solver to using SMP, instead of distributed parallel.

**Type**  
bool
Read Only  No

**UseLimitOnNumberOfCores**

Indicates whether to limit the number of processes for parallel execution.

Type  bool

Read Only  No

**SimulationSolutionSettings**

Mechanical specific simulation solution settings entity used for Journaling and Scripting. Not based on the common Infrastructure.Rsm.Queries entity.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type  string

Read Only  No

**Queue**

The queue that will be used on RSM task

Type  string

Read Only  Yes

**SolveManager**

Solve Manager used on RSM task.

Type  string

Read Only  Yes

**SolveProcessSetting**

Solve process setting used on RSM task.

Type  string

Read Only  No

**UpdateOption**

Used to specify the type of update, local or RSM.

Type  string
Read Only  No
Mesh

This container holds Mesh data for an instance of the Meshing application.

Methods

Edit

Opens the Meshing editor. If the editor has never been opened, the command will import the geometry file associated with the Geometry cell in the Mesh system.

Optional Arguments

Interactive

Specifies whether the editor should open in interactive mode. The default value is true.

Type: bool

Default Value: True

Example

To edit the mesh component with default optional parameter values:

```python
mesh.Edit()
```

Or by specifying optional parameter values:

```python
mesh.Edit(Interactive=True)
```

Exit

Exits the Meshing editor.

Optional Arguments

SaveDatabase

Bool flag to save the database on exit of the editor

Type: bool

Default Value: True
**Export**

Saves the current state of the Meshing editor and exports a *.meshdat file. If the Meshing editor is not currently open, the cached state (the state of the editor the last time it was closed) of the Meshing editor will be exported.

**Required Arguments**

**FilePath**  
The path and name of the .meshdat file to be written.

**Type**  
string

**ExportASMJournal**

Exports an ASM Journal (.wbjn) and supporting files for the model component.

**Required Arguments**

**FilePath**  
The path and name of the .wbjn file to be written. Supporting files written alongside.

**Type**  
string

**ExportGeometry**

Exports a PartManager database (.pmdb) file for the geometry in the model component.

**Required Arguments**

**FilePath**  
The path and name of the .pmdb file to be written.

**Type**  
string

**ExportMesh**

Exports a .acmo file for the mesh in the model component.

**Required Arguments**

**FilePath**  
The path and name of the .acmo file to be written.

**Type**  
string

**GetMechanicalMeshFile**

Returns the data reference to the container’s MechanicalMeshFile data entity.

**Return**  
A reference to the requested MechanicalMeshFile data entity.

**Type**  
DataReference

**GetMechanicalModel**

Returns the data reference to the container’s MechanicalModel data entity.

**Return**  
A reference to the requested MechanicalModel data entity.
**GetMechanicalSystemType**

Returns the data reference to the container's MechanicalSystemType data entity.

*Return*: A reference to the requested MechanicalSystemType data entity.

**GetMeshingImportOptions**

Query to return the data reference to the meshing import options

*Return*: The Data Entity containing settings for this component.

**Required Arguments**

- **Name**: The entity of interest.
  - *Type*: string

**Import**

Import a mesh data file. This action will delete the geometry container from the Mesh system and convert the Mesh cell into a file container. It is not possible to view meshes that are imported through this command in the Meshing editor; however, the files imported can be transferred to downstream mesh consumers (such as CFX, FLUENT, etc.) by way of normal component to component data transfer.

**Required Arguments**

- **FilePath**: The data file to be imported.
  - *Type*: string

- **MeshType**: The type of mesh data file.
  - *Type*: MeshFileType

**Example**

To import a data file into the mesh component with default optional parameter values:

```python
mesh.Import(filePath=r"C:\temp\data.cfx", meshType="CFX")
```

**ImportRepositoryMesh**

Import a mesh data file from a repository location. This action will delete the geometry container from the Mesh system and convert the Mesh cell into a file container. It is not possible to view meshes that are imported through this command in the Meshing editor; however, the files imported can be transferred to downstream mesh consumers (such as CFX, FLUENT, etc.) by way of normal component to component data transfer.
Required Arguments

**File**
The data file to be imported from the repository.

  **Type**  DataReference

**MeshType**
The type of mesh data file.

  **Type**  MeshFileType

---

**SendCommand**

Sends commands to the Meshing editor using JScript syntax.

If the Meshing editor is not open, then the editor's GUI will not be available to the script, causing some commands to fail. In this case, consider calling `Edit()` to open the editor before using `SendCommand`. Furthermore, if the Meshing editor is not open, `SendCommand` will start the editor without GUI, issue the specified command, and then close the editor. For multiple `SendCommand`s, this may degrade performance.

Required Arguments

**Command**
The command(s) to execute in the Meshing editor.

  **Type**  string

**Example**

To execute some arbitrary command (in this case, causing a dialog box to appear) in the Meshing editor:

```plaintext
mesh.SendCommand(Command=r"WBScript.Out("My Text",true);"
```

---

**Data Entities**

**GeneralMeshAssemblyProperties**

Class used to expose general properties for the mesh assembly workflow

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

  **Type**  string

  **Read Only**  No

**LengthUnit**

Target Length Unit for Assembled Model

  **Type**  string
**MechanicalModel**

The data entity that contains the identifiers which maintain a relationship between the Mesh data contain and the objects in the Meshing editor tree.

**Properties**

**Caption**

Caption to identify the model.

*Type*  
string

*Read Only*  
No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type*  
string

*Read Only*  
No

**ModelId**

Identifier which corresponds to the Model object in the Meshing editor tree. This ID can be used in conjunction with SendCommand to perform operations such as inserting mesh controls or manipulating mesh settings within the Meshing editor itself.

*Type*  
int

*Read Only*  
No

**Prototypel**

Identifier which corresponds to the Geometry object in the Meshing editor tree. This ID can be used in conjunction with SendCommand to perform operations such as inserting mesh controls or manipulating mesh settings within the Meshing editor itself.

*Type*  
int

*Read Only*  
No

**MechanicalSystemType**

This entity provides string based information about the physics, analysis, and solver settings for the Mesh component. As the Mesh component may be used independent of any specific physics, analysis, or solver, the properties exposed by this entity may take a value of "Any", meaning simply that it is up to the user to initialize and correctly configure the mesh settings within the Meshing editor.
Properties

**AnalysisTypeDisplayString**

The string which represents the current analysis type setting - this value will always be "Any".

Type: string
Read Only: Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string
Read Only: No

**PhysicsTypeDisplayString**

The string which represents current physics type setting - the value can be "Any" or "CFD".

Type: string
Read Only: Yes

**SolverTypeDisplayString**

The string which represents the current solver setting - the value can be "Any", "FLUENT", "CFX", or "POLYFLOW".

Type: string
Read Only: Yes

**MeshingImportOptions**

Import options for model assembly available on the downstream model

Properties

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string
Read Only: No

**NumberOfCopies**

By default this value is 0, specifies how many additional copies of the upstream mesh are needed

Type: int
**Read Only** No

**OriginX**
Displacement along the X axis

**Type** Quantity
**Read Only** No

**OriginY**
Displacement along the Y axis

**Type** Quantity
**Read Only** No

**OriginZ**
Displacement along the Z axis

**Type** Quantity
**Read Only** No

**Source**
The upstream mesh source

**Type** DataReference
**Read Only** No

**ThetaXY**
Rotation about the XY plane

**Type** Quantity
**Read Only** No

**ThetaYZ**
Rotation about the YZ plane

**Type** Quantity
**Read Only** No

**ThetaZX**
Rotation about the ZX plane

**Type** Quantity
Mesh

Read Only  No
Microsoft Office Excel Analysis

This container holds information to expose data from an instance of Microsoft Excel as Workbench parameters.

**Methods**

*GetExcelSetup*

Get the DataReference of the MSExcelSetup entity. There is only one MSExcelSetup entity per Analysis container. An exception is thrown if the entity is not found.

**Return**

The DataReference of the MSExcelSetup entity.

**Type** DataReference

**Example**

The following example shows how the user can get a setup entity to set a different NamedRangeKey value.

```plaintext
system1 = GetSystem(Name="XLS")
analysis1 = system1.GetContainer(ComponentName="Analysis")
setup1 = analysis1.GetExcelSetup()
setup1.NamedRangeKey = "MyPrefix"
```

**Data Entities**

*MSExcelFile*

This entity represents an Excel file added to the Analysis container in order to expose data as Workbench parameters and perform calculations based on parameters. It is created by the MSExcelSetup.AddFile method which copies and registers the original user's file into the project files.

**Properties**

*DisplayText*

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

**Read Only** No
**ErrorMessage**
Error message if the calculation failed.

Type: string
Read Only: No

**FileName**
Name of the file.

Type: string
Read Only: No

**MacroName**
Name of the macro used to calculate the workbook.

Type: string
Read Only: No

**OriginalFilePath**
Original Path of the file.

Type: string
Read Only: No

**State**
Calculation state of the file.

Type: Status
Read Only: No

**UseMacro**
Set to True if a Visual Basic macro is used to calculate the workbook.

Type: bool
Read Only: No

**Methods**

**GetRange**
Gets the DataReference of an MSExcelRange entity from its name. An exception is thrown if the entity is not found.

Return: The DataReference of the MSExcelRange entity.
**Type**  
DataReference

**Required Arguments**

**Name**  
The Name of the MSExcelRange to retrieve.

  **Type**  
string

**Example**

The following example shows how the user can get an MSExcelRange named "WB_Thickness".

```plaintext
system1 = GetSystem(Name="XLS")
analysis1 = system1.GetContainer(ComponentName="Analysis")
setup1 = analysis1.GetExcelSetup()
file1 = setup1.GetFile()
rangel = file1.GetRange(Name="WB_Thickness")
write rangel.CellRange
write rangel.Value
```

**GetRanges**

Gets the list of all MSExcelRange entities.

**Return**  
The DataReferenceSet containing all the MSExcelRange entities associated with the MSExcelFile.

  **Type**  
DataReferenceSet

**Example**

The following example shows how the user can get all the MSExcelRange entities of an MSExcelFile.

```plaintext
system1 = GetSystem(Name="XLS")
analysis1 = system1.GetContainer(ComponentName="Analysis")
setup1 = analysis1.GetExcelSetup()
file1 = setup1.GetFile()
ranges = file1.GetRanges()
```

**PublishParameter**

Publishes an MSExcelRange as a Parameter in the Workbench project. The IsOutput argument allows to publish the range as an input or an output parameter. The method returns the created Parameter entity.

**Return**  
DataReference of the created parameter.

  **Type**  
DataReference

**Required Arguments**

**ExcelRange**  
The MSExcelRange to be published as a parameter.

  **Type**  
DataReference

**Optional Arguments**

**IsOutput**  
True to specify that the range is published as an output parameter, or false for input parameter.
**Type**  
`bool`

**Default Value**  
`False`

**Example**  
The following example shows how to publish ranges as input or output parameters.

```csharp
system1 = GetSystem(Name="XLS")
analysis1 = system1.GetContainer(ComponentName="Analysis")
setup1 = analysis1.GetExcelSetup()
file = setup1.AddFile(filePath="C:\Test\ProcessCalculation.xlsx")
ValvePosition = file.GetRange(name="WB_Valve_Position")
T1 = file.GetRange(name="WB_T1")
input1 = file.PublishParameter(ValvePosition)
output1 = file.PublishParameter(T1)
```

**Reload**  
Reloads an MSExcelFile to synchronize the data between the Microsoft Office Excel application and Workbench. For instance, it is necessary to call this method when the ExcelSetup.NamedRangeKey has been modified, in order to filter the named ranges based on the new NamedRangeKey's value. It is also necessary to reload the file if the workbook has been edited outside of Workbench, for instance to change a formula. The method recreates MSExcelRange and Parameter entities as required. The existing results in Workbench are invalidated.

**Example**  
The following example shows how to reload a file.

```csharp
system1 = GetSystem(Name="XLS")
analysis1 = system1.GetContainer(ComponentName="Analysis")
setup1 = analysis1.GetExcelSetup()
file = setup1.GetFile()
file.Reload()
```

**UnpublishParameter**  
Unpublishes an MSExcelRange as a Parameter from the Workbench project, which means deleting the Parameter entity that was created in association to this MSExcelRange.

**Required Arguments**

**ExcelRange**  
The MSExcelRange to unpublish.

**Type**  
`DataReference`

**Example**  
The following example shows how to publish and unpublish ranges as input or output parameters.

```csharp
system1 = GetSystem(Name="XLS")
analysis1 = system1.GetContainer(ComponentName="Analysis")
setup1 = analysis1.GetExcelSetup()
file = setup1.AddFile(filePath="C:\Test\ProcessCalculation.xlsx")
ValvePosition = file.GetRange(name="WB_Valve_Position")
input1 = file.PublishParameter(ValvePosition)
output1 = file.UnpublishParameter(ValvePosition)
```

---

*Microsoft Office Excel Analysis*

Release 15.0 - © SAS IP Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.*
**MSEExcelRange**

This entity represents an Excel named range that matches the prefix string defined by the Parameter Key. The MSEExcelRange entities are created automatically when an Excel file is added, or reloaded, by filtering all the named ranges found in the Excel workbook with the Parameter Key. The MSEExcelRange is not published as a parameter by default: use the MSEExcelFile.PublishParameter method to expose the range as an input or output parameter in the Workbench project. The named ranges can contain a single cell for the value, or two cells for the value and the unit string.

**Properties**

**CellRange**

The coordinates defining the Excel range in the workbook.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**Value**

The current value of the range.

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**ValueQuantityName**

The quantity name.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**Methods**

**GetParameter**

Gets the Parameter entity associated with a published MSEExcelRange entity. If the MSEExcelRange is not published as a parameter, the method returns null.

| Return     | The Parameter associated with the MSEExcelRange, or null if it is not published. |
**DataReference**

**Example**
The following example shows how the user can get the parameter associated with a range.

```python
system1 = GetSystem(Name="XLS")
analys1 = system1.GetContainer(ComponentName="Analysis")
setup1 = analys1.GetExcelSetup()
file1 = setup1.GetFile()
rang1 = file1.GetRange(Name="WB_Thickness")
parameter1 = rang1.GetParameter()
write parameter1.Name
```

**MSExcelSetup**

This entity holds properties to setup the data exchange with the Microsoft Office Excel application and information about the state of the connection with the instance of Microsoft Office Excel. There is one unique MSExcelSetup entity instance per Analysis container.

**Properties**

**DisplayText**
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type** string
- **Read Only** No

**ExcelStatus**

Status of the connection with the instance of the Microsoft Office Excel application.

- **Type** ExcelConnectionState
- **Read Only** Yes

**ExcelVersion**

Version number of the Microsoft Office Excel instance connected with Workbench.

- **Type** string
- **Read Only** Yes

**NamedRangeKey**

The Named Ranges Key is a prefix string used to filter the named ranges found in the Excel workbook. The default value is set using the value of the related user preference. It is possible to use an empty string in order to retrieve all named ranges.

- **Type** string
- **Read Only** No
**UnitSystemName**

The name of the unit system used for the Analysis container.

**Type** string

**Read Only** No

**Methods**

**AddFile**

Adds a file to an MSExcelSetup entity by providing its FilePath. The method copies and registers the file into the Workbench project and returns an MSExcelFile entity if successful. The method also creates an MSExcelRange entity for each named range matching the ExcelSetup.NamedRangeKey prefix string ("" by default). If the file does not exist, is not an Excel file or cannot be registered into the project, the method throws an exception. If a file was already added to the MSExcelSetup, the method throws an exception as well because only one file can be handled in each Analysis container. To use another file, the user has to Reset the data container first.

**Return** The DataReference of the added file.

**Type** DataReference

**Required Arguments**

**FilePath** The Path of the original Microsoft Office Excel file to add.

**Type** string

**Example**
The following example shows how to add a file to the MSExcelSetup entity.

```python
system1 = GetSystem(Name="XLS")
analysis1 = system1.GetContainer(ComponentName="Analysis")
setup1 = analysis1.GetExcelSetup()
file = setup1.AddFile(FilePath="C:\Test\ProcessCalculation.xlsx")
write file.FileName
ranges = file.GetRanges()
```

**DeleteFile**

Deletes a file from an MSExcelSetup entity and from the Workbench project file management. All the MSExcelRange entities and associated Parameter entities are deleted as well.

**Required Arguments**

**File** The DataReference of the MSExcelFile entity to be deleted.

**Type** DataReference

**Example**
The following example shows how to delete a file from an MSExcelSetup entity.

```python
system1 = GetSystem(Name="XLS")
```
analysis1 = system1.GetContainer(ComponentName="Analysis")
setup1 = analysis1.GetExcelSetup()
file = setup1.GetFile()
setup1.DeleteFile(file)

GetFile

Gets the MSExcelFile entity associated to the MSExcelSetup entity. If the MSExcelSetup entity has no file, the method returns null. If the optional argument Name is specified but does not correspond to a file associated to the MSExcelSetup, the method throws an exception.

Return

The DataReference of the retrieved file entity.

Type

DataReference

Optional Arguments

Name

Name of the MSExcelFile entity to retrieve.

Type

string

Example

The following example shows how the user can get a file entity from a setup entity to retrieve one of its properties.

system1 = GetSystem(Name="XLS")
analysis1 = system1.GetContainer(ComponentName="Analysis")
setup1 = analysis1.GetExcelSetup()
file1 = setup1.GetFile()
write file1.Name
Parameters

This container holds project-level Parameters and Design Points.

Methods

GetAllParameters

Returns the set of all parameters associated with all entity properties in a given container. Parameters not associated with containers are not returned.

Return

The set of parameters present in the given container.

Type     DataReferenceSet

Example

In this example 'paramSet' becomes the set of parameters associated with the properties of any entity that resides within the Results container of system1.

```plaintext
results1 = system1.GetContainer(ComponentName="Results")
paramSet = results1.GetAllParameters()
```

Data Entities

DesignPoint

The data entity which describes a project-level design point.

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type     string
Read Only     No

Exported

Specifies whether the design point is exported.

Type     bool
Parameters

**Read Only**  No

**HasValidRetainedData**

Indicates whether the data model and files for this design point is retained in the project and valid. This can happen for a retained DP; or a non-retained DP before previously retained data is deleted.

**Type**  bool

**Read Only**  Yes

**IsUpToDate**

True if the design point is up to date. False if the design point requires an update.

**Type**  bool

**Read Only**  Yes

**Note**

Contains user-defined notes about the design point.

**Type**  string

**Read Only**  No

**Retained**

Gets or sets whether the design point is retained.

**Type**  bool

**Read Only**  No

**UpdateOrder**

The update order of the design point.

**Type**  double

**Read Only**  No

**Methods**

**AreParameterValueEqual**

Comparison to see if all parameter values within two different design points are equivalent.

**Return**  True if all parameter values are equal.

**Type**  bool

**Required Arguments**

**DesignPoint2**  The second design point
**Type** DataReference

**CopyParameterExpressions**
Copies all input parameter expressions from one design point to another.
Invalidates all output parameter values in the destination design point.

**Required Arguments**

**ToDesignPoint** The destination design point.

**Delete**
Deletes a design point, the associated directory, and all design point files from the project.
Note that the active design point cannot be deleted.

**Duplicate**
Creates a new design point in which all parameters have the same values as in the specified original design point.

**Return** The created design point entity.

**GetType**
Returns the value of a specified parameter in a given design point.

**Return** The value of the parameter in the specified design point.

**Required Arguments**

**Parameter** The parameter data reference.

**SetParameterExpression**
Sets the expression of the specified input parameter in the given design point.

**Required Arguments**

**Expression** The string with the expression for the parameter.

**Type** string

**Parameter** The input parameter data reference.

**Type** DataReference
Example
The following example illustrates the setting of a parameter expression for a given design point.

```csharp
dp = Parameters.GetDesignPoint("0")
param = Parameters.GetParameter("P1")
dpsetParameterExpression(param, "cos(1)")
```

**SetParameterExpressions**
Sets expressions of the specified parameters in the specified design point.

**Required Arguments**

**ParameterExpressions**
The parameters and expressions

*Type* `IDictionary<DataReference, string>`

**Parameter**
The data entity which describes a project-level Parameter.

**Properties**

**Description**
A human-readable description of the parameter.

*Type* `string`

*Read Only* No

**DisplayText**
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type* `string`

*Read Only* No

**ErrorMessage**
An error message as a result of a validation or expression evaluation error.

*Type* `string`

*Read Only* Yes

**Expression**
The parameter definition as an expression. This property may be a constant expression, or it may depend on the values of other parameters.

*Type* `string`
Read Only No

**ExpressionType**

Indicates whether the expression is constant, derived from another expression, or undefined.

**Type** ExpressionType

Read Only Yes

**Usage**

Specifies how the parameter is used in the data model.

**Type** ParameterUsage

Read Only Yes

**Value**

The current value of the parameter. For derived parameters, this value is the evaluated expression result.

**Type** Object

Read Only Yes

**ValueQuantityName**

Gets the value quantity name if ValueSpec is a QuantitySpec. Null otherwise.

**Type** string

Read Only No

**Methods**

**Delete**

Deletes a parameter.

A parameter can only be deleted when it is no longer associated with any properties/entities in the project.

**Disassociate**

Disassociates a data model property from an existing parameter.

To associate the parameter with a property, call AssociateParameter.

**Required Arguments**

**Entity**

The data model entity that holds the property to be disassociated.

**Type** DataReference

**PropertyName**

The name of the property to be disassociated.
**DisassociateInContext**

Disassociates a data model property for a given context from an existing parameter.

To associate the parameter with a property in context, call AssociateParameterInContext.

**Required Arguments**

- **Entity**
  
  The data model entity that holds the property to be disassociated.

  **Type** DataReference

- **PropertyName**
  
  The name of the property to be disassociated.

  **Type** string

**Optional Arguments**

- **Context**
  
  The context of the association.

  **Type** DataReference

**Example**

The following example illustrates proper DisassociateParameterInContext invocation:

```python
entity1 = ...
parameter1 = Parameters.GetParameter(Name="p1")
parameter1.DisassociateParameterInContext(entity1, "SampleProperty")
```

Callers may provide a Context argument.

```python
contextEntity1 = ...
parameter1.DisassociateParameterInContext(entity1, "SampleProperty", contextEntity1)
```

**GetAssociatedEntityProperties**

Returns the entity properties associated with a given parameter. If a container is specified, returns only the entity properties for the given container.

**Return**

A ParameterizedEntityPropertiesCollection dictionary containing a data reference to each entity with parameterized properties and the list of property names within that entity.

**Type** ParameterizedEntityPropertiesCollection

**Optional Arguments**

- **Container**
  
  Optionally specifies a container for which the entity properties should be returned.

  **Type** DataContainerReference
**GetDependencies**

For a derived parameter, returns the list of parameters referred to by this parameter's expression.

**Return**

Data references to the parameters that this parameter is dependent upon.

**Type**  List<DataReference>

**SetQuantityName**

Sets a parameter's quantity name.

**Required Arguments**

**QuantityName**  The quantity name.

**Type**  string

**SetQuantityUnit**

Set the unit string for a parameter across all design points.

**Required Arguments**

**Unit**  The new units of the parameter.

**Type**  string

**ParameterSummaryChart**

This is a summary chart that will display all parameters against all design points. This is useful for understanding and visualizing the full parameter space, but is not very useful for comparing one parameter against another. For parameter vs. parameter charts, see CreateParameterVsParameterChart.

**Properties**

**Chart**

This stores the data reference to any generated charts from the Graphics system. e.g. a result of CreateMultiAxisChart.

**Type**  DataReference

**Read Only**  No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  string

**Read Only**  No
**IsBaseDesignPointExcluded**

If this property is set the base design point will be excluded.

**Type**  
`bool`

**Read Only**  
No

**Variables**

The variables that have been added to the chart.

**Type**  
`DataReferenceSet`

**Read Only**  
No

**Methods**

**Delete**

Deletes a parameter chart.

**ParameterVsParameterChart**

This chart type can be used to plot one parameter against another or parameters vs. design points.

There are 4 accessible axes, X-Top, X-Bottom, Y-Left and Y-Right.

```csharp
chart1 = Parameters.CreateParameterVsParameterChart();
chart1.XAxisBottom = Parameters.GetParameter("P1");
chart1.XAxisTop = Parameters.GetParameter("P2");
chart1.YAxisLeft = Parameters.GetParameter("P3");
chart1.YAxisRight = Parameters.GetParameter("P4");
```

**Properties**

**Chart**

This stores the data reference to any generated charts from the Graphics system. e.g. a result of CreateChartXY.

**Type**  
`DataReference`

**Read Only**  
No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
`string`

**Read Only**  
No
IsBaseDesignPointExcluded

If this property is set the base design point will be excluded.

Type       bool
Read Only  No

XAxisBottom

The parameter entity to be plotted on the primary (bottom) x-axis.

Type       DataReference
Read Only  No

XAxisTop

The parameter entity to be plotted on the secondary (top) x-axis.

Type       DataReference
Read Only  No

YAxisLeft

The parameter entity to be plotted on the left (primary) y-axis.

Type       DataReference
Read Only  No

YAxisRight

The parameter entity to be plotted on the right (secondary) y-axis.

Type       DataReference
Read Only  No

Methods

Delete

Deletes a parameter chart.
Polyflow

Polyflow Setup
This container holds Setup data for an instance of Polyflow.

Methods

AddFile
Allows to import any kind of file into the Inputs sub-directory of the current Polyflow system

```python
setup1.AddFile(FilePath="D:/temp/viscosity.crv")
```
This command will copy the viscosity.crv file from the D:\temp directory into the Inputs directory of the current Polyflow system.

**Required Arguments**

FilePath Input : Path of file to be added.

**Type** string

Edit
Starts the editor (Polydata) given a Polyflow Setup Container

```python
setup1.Edit()
```
This command will launch the editor (Polydata) of the selected Setup cell.

Exit
Stops the editor (Polydata) or the solver (Polyflow) given a Polyflow Setup or Solution Container

```python
setup1.Exit()
```
This command will stop the editor (Polydata) of the selected Setup cell.

```python
solver1.Exit()
```
This command will stop the solver (Polyflow) of the selected Solution cell.

GetPolyflowMesh
Returns the PolyflowMesh object stored in the Polyflow Setup component.

**Return** The PolyflowMesh object stored in the Polyflow Setup component.

**Type** DataReference
**ImportMesh**

Allows to import a mesh file (gambit *.neu file, *.poly file, *.msh Fluent Mesh file or Polyflow Mesh file). For *.neu and *.poly and a *.msh fluent file, a mesh conversion is automatically performed to get eventually a Polyflow Mesh file (*.msh)

```python
setup1.ImportMesh(FilePath="D:/temp/swell.msh")
```

This command will copy the swell.msh file from the D:\temp directory into the directory of the current Polyflow system.

**Required Arguments**

- **FilePath**
  - Input: path of mesh file to be imported (*.neu, *.poly or *.msh)
  - Type: string

**ImportSetup**

Allows to import a setup file (*.dat file) into the current Polyflow system

```python
setup1.ImportSetup(FilePath="D:/temp/swell.dat")
```

This command will copy the swell.dat file from the D:/temp directory into the directory of the current Polyflow system.

**Required Arguments**

- **FilePath**
  - Input: Path of Polyflow setup file to be imported.
  - Type: string

**ImportUDF**

Allows to import the udf file into the Inputs sub-directory of the current Polyflow system

```python
setup1.ImportUDF(UdfFilePath="D:/temp/swell.udf")
```

This command will copy the swell.udf file from the D:/temp directory into the Inputs directory of the current Polyflow system.

**Required Arguments**

- **UdfFilePath**
  - Input: Path of Polyflow udf file to be imported
  - Type: string

**SetMeshPreferences**

Allows to specify which mesh of upstream polyflow system will be used in the current polyflow system. One does not provide a mesh file name, but a specific key to the corresponding mesh. The Mesh restart mode can take two values: "NoUpstreamMeshFile" or "SingleMeshFile"

```python
setup1.SetMeshPreferences(
  meshRestartMode="SingleMeshFile",
  meshKey="mesh (t=0.5000000E+00)[formatted]")
```

This command will take the mesh generated by the upstream polyflow system at time t=0.5 (and that is formatted)
Note: the mesh keys can be found in the "last_result.pub" file generated by the Polyflow solver in the upstream system!

**Required Arguments**

**meshKey**
Mesh key (among the list of available meshes created by an upstream polyflow system)

*Type*  string

**meshRestartMode**
Mesh restart mode

*Type*  MeshRestartMode

**StartFuseTool**

Starts Polyfuse given a Polyflow Setup Container

```java
setup1.StartFuseTool()
This command will launch the mesh manipulation tool (Polyfuse) on the selected Setup cell.
```

**StartMaterialsTool**

Starts Polymat given a Polyflow Setup Container

```java
setup1.StartMaterialsTool()
This command will launch the material parameters fitting tool (Polymat) on the selected Setup cell.
```

**StartPreferences**

Starts the Editor of Polydata preferences given a Polyflow Setup Container or Starts the Editor of Polyflow preferences given a Polyflow Solution Container

This command applies to Setup and Solution cells.
```java
setup1.StartPreferences()
solution1.StartPreferences()
This command will launch the Preferences setting tool (Polypref) of the selected Setup cell.
This command will launch the Preferences setting tool (Polypref) of the selected Solution cell.
```

**Data Entities**

**PolydataPreference**

Contains a reference to the preferences file for Polydata (setup editor). It is a provider of the preferences file.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type*  string

*Read Only*  No
**File**

Data reference to the preferences file for Polldata.

**PolyflowMesh**

Contains a reference to a Polyflow mesh file. It is a provider for a Polyflow mesh file.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**File**

Data reference to the mesh file

**PolyflowSetup**

Contains a reference to a Polyflow data file. It is a provider of the data file (containing the setup).

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**File**

Data reference to the Polyflow Dat file
Polyflow Solution

This container holds Solution data for an instance of Polyflow.

Methods

ClearTemporaryFiles

Clear temporary files generated by Polyflow solver (if the solver is not running!) given a Polyflow Solution Container

```plaintext
solution1.ClearTemporaryFiles()
```

Exit

Stops the editor (Polydata) or the solver (Polyflow) given a Polyflow Setup or Solution Container

```plaintext
setup1.Exit()
This command will stop the editor (Polydata) of the selected Setup cell.

solver1.Exit()
This command will stop the solver (Polyflow) of the selected Solution cell.
```

GetComponentSettingsForRsmDpUpdate

This query is used to obtain the ComponentSettingsForRsmDpUpdate object for Journaling and Scripting

GetPolyflowSolution

Returns the PolyflowSolution object stored in the Polyflow Solution component.

**Return**

The PolyflowSolution object stored in the Polyflow Solution component.

**Type** DataReference

GetSolutionSettings

This query is used to obtain the solution settings object for Journaling and Scripting

SetResultsPreferences

Allows to specify which result (res/rst/csv) of upstream polyflow system will be used in the current polyflow system: one does not provide filenames, but specific keys to the corresponding result files.

The solver restart mode can take the following values:

"No initialization from upstream system",
"Restart with a single Polyflow results file",
"Restart with Polyflow results and restart files",
"Restart with a single Polyflow CSV file",
"Restart with Polyflow CSV and restart files", 

"Restart with a list of Polyflow results files (for transient mixing tasks only!)",
"Restart with a list of Polyflow CSV files (for conversion of a list of Polyflow CSV files)"

Note: the result (res, rst, csv) keys can be found in the "last_result.pub" file generated by the Polyflow solver in the upstream system!

**Required Arguments**

- `csvKey`  
  Csv key (among the list of available csv created by an upstream polyflow system)  
  **Type**  
  string

- `restartKey`  
  Restart key (among the list of available restart created by an upstream polyflow system)  
  **Type**  
  string

- `resultKey`  
  Result key (among the list of available results created by an upstream polyflow system)  
  **Type**  
  string

- `solverRestartMode`  
  Solver restart mode  
  **Type**  
  `SolverRestartMode`

**Example**

```python
solution1.SetResultsPreferences(
    solverRestartMode="Restart with a single Polyflow CSV file",
    resultKey="undefined",
    restartKey="undefined",
    csvKey="csv (t=0.500000E+00)[formatted]"
)
```

This command will take the csv generated by the upstream polyflow system at time t=0.5 (and that is formatted)

**StartCurveTool**

Starts Polycurve (curves viewer) given a Polyflow Solution Container

```python
solution1.StartCurveTool()
```

This command will launch Polycurve of the selected Solution cell.

**StartDiagnosticsTool**

Starts Diagnostics tool (Polydiag) given a Polyflow Solution Container

```python
solution1.StartDiagnosticsTool()
```

This command will launch the Diagnostics tool (Polydiag) of the selected Solution cell.

**StartListingViewer**

Starts the listing viewer given a Polyflow Solution Container
solution1.StartListingViewer()
This command will launch the listing viewer tool (Polyist) on the selected Solution cell.

**StartPreferences**

Starts the Editor of Polydata preferences given a Polyflow Setup Container or Starts the Editor of Polyflow preferences given a Polyflow Solution Container

**setup1.StartPreferences()**
This command applies to Setup and Solution cells.

**solution1.StartPreferences()**
This command will launch the Preferences setting tool (Polypref) of the selected Solution cell.

**StartStatisticsTool**

Starts Polystat given a Polyflow Solution Container

**solution1.StartStatisticsTool()**
This command will launch the statistical analyzer tool (Polystat) on the selected Solution cell.

**SwitchToBackgroundMode**

Switch the Update in progress into background mode. This will enable operations that are not allowed during an Update in foreground mode (e.g. Project Save).

This command is not normally useful in a script. Journals may record the invocation of this command after an Update invoke, as the result of GUI activity while the Update is in progress. However, replay of these journals will always wait for the Update invoke to complete before invoking the next command, rendering this step ineffectual.

**Data Entities**

**PolyflowPreference**

Contains a reference to the preferences file for Polyflow (solver). It is a provider of the preferences file.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string
**Read Only** No

**File**

Data reference to the preferences file for Polyflow.

**Type** DataReference
PolyflowSolution

Contains a reference to a Polyflow listing file. It is a provider of the listing file (containing a summary of the simulation).

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type string

Read Only No

File

Data reference to the listing file generated by the Polyflow solver and containing a summary of the simulation.

Type DataReference

Read Only No
Project

This container holds the Systems, Components and Templates in the project.

Data Entities

Component

Data entity representing a component of a system in the schematic.

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string
Read Only: No

Notes

A note about the component.

Type: string
Read Only: No

Methods

Clean

Delete the heavyweight data (e.g. solution and/or results) of this component to reduce the size of the project.

DeleteShare

Removes all sharing affecting the specified component.

Optional Arguments

System

The target system holding the shared component. Used if the component is shared in more than one system.
**DeleteTransfer**

Deletes the transfer connection between two components.

**Required Arguments**

- **TargetComponent**: A reference to the component that was receiving data.
  - **Type**: DataReference

**GetContainer**

Query to return the Container for the component.

- **Return**: Container Reference
  - **Type**: DataContainerReference

**Refresh**

Refreshes the input data for a component by reading all changed data from upstream (source) components. Does not perform any calculations or updates based on the new data.

**RemoveFromSystem**

Deletes the component and all dependents from a system.

**Required Arguments**

- **System**: A reference to the system containing the component to be deleted. This is needed in case the component is shared between more than one system.
  - **Type**: DataReference

**ReplaceWithShare**

Replaces a component in a system with a shared copy of a component from another system. The data sources of any downstream components are updated to include the new component.

**Required Arguments**

- **ComponentToShare**: The component to share into the target system.
  - **Type**: DataReference

**Optional Arguments**

- **SourceSystem**: The system from which to share the specified component.
  - **Type**: DataReference
- **TargetSystem**: The target system into which to share the component.
  - **Type**: DataReference
Example
The following example illustrates a replace of a component with a shared copy of another system's component.

```csharp
componentToShare = sys2.GetComponent("componentName")
componentToReplace = sys1.GetComponent("componentName")
componentToReplace.ReplaceWithShare(ComponentToShare=componentToShare)
```

Note that you do not need to specify the TargetSystem and SourceSystem. They are only retained for legacy compatibility.

Reset
Resets the component by removing all user input and result data.

TransferData
Creates a data transfer connection between two components in the project.

Required Arguments
TargetComponent A reference to the component receiving data.
Type DataReference

TransferSpecificData
Create a data transfer connection between two existing components, specifying the type of data to be transferred. This is used in some situations where more than one type of data can be exchanged between two components.

See also component.TransferData(TargetComponent=...)

Required Arguments
TargetComponent A reference to the component receiving data.
Type DataReference
TransferDataName The name of the type of data to be transferred between components.
Type string

Update
Updates the component by refreshing the input from all upstream components and then performs a local calculation based on current data.

Optional Arguments
AllDependencies If true, also updates all upstream dependencies of this component.
Type bool
**ComponentTemplate**

This entity provides information used in the creation of a new component in the project. A component template has no user modifiable properties and is typically only referenced when creating a new system that includes data connections.

**Properties**

**DisplayName**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**    string
**Read Only**  No

**Methods**

**CreateComponent**

Creates a component based on a given component template.

**Required Arguments**

**System**    The system to which the created component should be added.

**Type**    DataReference

**Optional Arguments**

**AllSystemProperties**    The system properties to provide to the container creation command.

**Type**    SystemPropertyDictionary

**CreatedComponent**    The created component.

**Type**    Output<DataReference>

**Name**    The name to use for the created component.

**Type**    string

**UpstreamComponent**    A list of upstream components which will provide transfer data to the created component.

**Type**    List<DataReference>

**Example**

The following example illustrates Component creation from a template.

```plaintext
system1 = CreateSystem(...)
template = GetComponentTemplate("MyTemplate")
template.createComponent(System=system1)
```
Optional properties allow you to modify component creation, including the establishment of upstream component connections.

```python
system1 = CreateSystem(...)
upstreamComponent = ...
template = GetComponentTemplate("MyTemplate")
template.createComponent(Name="MyComponent", System=system1, UpstreamComponent=[upstreamComponent], AllSystemProperties=None)
```

**CustomProjectTemplate**

A project snippet template containing multiple systems and their links.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type** string
- **Read Only** No

**SystemNames**

The names of the systems to create, one for each entry in the SystemTemplates list.

- **Type** List<string>
- **Read Only** No

**SystemTemplates**

Templates for the systems making up this snippet.

- **Type** List<DataReference>
- **Read Only** No

**Methods**

**CreateProject**

Instantiates all the systems contained in a project snippet template.

**Delete**

Deletes the project template from the toolbox and user’s Workbench application data.

**SchematicSettings**

This data entity holds properties that control the appearance of the Project Schematic.
Properties

DisplayText
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type string
Read Only No

Notes
A note about the project.

Type string
Read Only No

System
A collection of components brought together to complete a specific type of analysis or data generation within a project.

Properties

AnalysisType
The analysis type associated with this system.

Type string
Read Only No

Components
The set of components which form this system.

Type List<DataReference>
Read Only No

DisplayText
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type string
Read Only No

Notes
A note about the system.

Type string
Read Only  No

Physics
The physics associated with this system.
Type  List<string>
Read Only  No

Solver
The solver associated with this system.
Type  List<string>
Read Only  Yes

Methods
Delete
Deletes the system and all contained data from the project.

Duplicate
Creates a new system containing a copy of all the data in this system.
Return  The reference to the created system.
Type  DataReference

Optional Arguments
ComponentsToShare  The components to share into the new system. Used if the system being duplicated has shared components, and we wish to continue using those shares in the new system.
Type  List<DataReference>

Name  The name for the new system.
Type  string

Position  Type of positioning of the new system.
Type  PositionType

RelativeTo  The system to which new system is positioned relative to.
Type  DataReference

Example
The following example illustrates System duplication. No components are shared. The new, copied system will appear to the right of the existing system. Its name will be "MyNewSystme".
mySystem = CreateSystem(...)  
mySystem2 = mySystem.Duplicate(ComponentsToShare=[],  
Position="Right",  
RelativeTo=mySystem,  
Name="MyNewSystem")

**GetComponent**

Query to return the component reference for a given component base name in the system.

**Return**  
Data Container Reference

**Type**  
DataReference

**Required Arguments**

**Name**  
Base Name of the Component

**Type**  
string

**GetContainer**

Query to return the container for a given component base name in the system.

**Return**  
Data Container Reference

**Type**  
DataContainerReference

**Required Arguments**

**ComponentName**  
Base Name of the Component

**Type**  
string

**GetReplaceableTemplates**

A query to return the list of templates that can be used to replace the current system.

See also template.ReplaceSystem(System=...).

**Return**  
The list of System Template references that can be used to replace this system.

**Type**  
List<DataReference>

**Move**

Moves an existing system in the schematic.

**Required Arguments**

**Position**  
Type of positioning.

**Type**  
PositionType

**RelativeTo**  
The system that the moved system is positioned relative to.
**Type** DataReference

**RecreateDeletedComponents**
Recreates new versions of any components that have been deleted from the system.

**Refresh**
Refreshes the input data for all components in the system by reading changed data from upstream sources. Does not perform any calculations or updates based on the new data.

**Update**
Updates all components in the system by refreshing the input from all upstream sources, and then performing local calculation based on current data.

**Optional Arguments**

**AllDependencies**  If true, also updates all upstream dependencies of this system.

**Type** bool

**Template**
A template that is used to create a system within the project.

**Properties**

**AnalysisType**
The analysis type associated with this system template.

**Type** string

**Read Only** No

**DirectoryName**
The user-visible name for the type of systems created from this template.

**Type** string

**Read Only** No

**DisplayText**
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

**Read Only** No

**Physics**
The physics associated with this system template.
**Solver**

The solver associated with this system template.

- **Type**: List<string>
- **Read Only**: No

**SystemType**

The system type, displayed in the User Interface in the header of the system block.

- **Type**: string
- **Read Only**: No

**SystemTypeAbbreviation**

The system type abbreviation, used to create the system unique directory name and also serves as the system object’s base name.

- **Type**: string
- **Read Only**: No

**Methods**

**CreateSystem**

Creates a new system from a system template.

- **Return**: The reference to the created system.
  - **Type**: DataReference

**Optional Arguments**

**ComponentsToShare**

A list of components to share into the newly created system.

- **Type**: List<DataReference>

**DataTransferFrom**

A list of data transfer specifications to transfer data into the new system. This is used to create follow-on systems. This and DataTransferTo are mutually exclusive.

- **Type**: List<TransferDataToNewComponentSpec>

**DataTransferTo**

A list of data transfer specifications to transfer data from the new system into existing components. This is used to create a preceding system. This and DataTransferFrom are mutually exclusive.

- **Type**: List<TransferDataFromNewComponentSpec>
**Name**
The name for the new system.

**Type**
string

**Position**
The type of positioning of the new system.

**Type**
PositionType

**Default Value**
Default

**RelativeTo**
The system to which the new system is positioned relative to.

**Type**
DataReference

**Example**
Two examples are presented. This first section gets the template for a "Fluid Flow (CFX)" system, and then creates an empty, stand-alone system in the default position.

```python
template1 = GetTemplate(TemplateName="Fluid Flow", Solver="CFX")
system1 = template1.CreateSystem()
```

In the second part of the example, we create a Static Structural (ANSYS) system connected to the previous Fluid Flow system. The Geometry component from the Fluid Flow system (component1) is shared with the new Static Structural system. The Solution component from the Fluid Flow system (component2) transfers results data into the Setup component of the Static Structural system. The new system is created to the right of the existing system.

```python
template2 = GetTemplate(TemplateName="Static Structural", Solver="ANSYS")
component1 = system1.GetComponent(Name="Geometry")
component2 = system1.GetComponent(Name="Solution")
componentTemplate1 = GetComponentTemplate(Name="SimulationSetupTemplate_StructuralStaticANSYS")
system2 = template2.CreateSystem(
    CellsToShare=[component1],
    DataTransferFrom=[
        "FromComponent": component2,
        "TransferName": "CFXTransferResultsTemplate",
        "ToComponentTemplate": componentTemplate1],
    Position="Right",
    RelativeTo=system1)
```

**ReplaceSystem**

Replace an existing system with one based on the selected System Template.

There are a number of restrictions on the template that can be used to replace an existing system. See system.GetReplaceableTemplates() to determine the list of allowable templates.

**Return**
The reference to the created system.

**Type**
DataReference

**Required Arguments**

**System**
The existing system to replace.

**Type**
DataReference
Optional Arguments

Name  The name for the new system.
  Type  string

Project File Types

This container holds File Types registered with the File Manager.

Data Entities

FileType

FileType defines file type information including a short description.

Properties

Description

The description of the file type.

Type  string
Read Only  Yes

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type  string
Read Only  No

ExtensionPatterns

The possible file extension patterns to match in a regular expression. The patterns are used to check whether this file type can handle a set of file extensions.

Type  List<string>
Read Only  Yes

Extensions

The possible file extensions (with the leading "."). These extensions can be displayed to the user in a file dialog filter, for example.

Type  List<string>
Read Only  Yes
**Project Files**

This container holds Files registered with the File Manager.

**Methods**

**GetFiles**

Gets all the File References that are associated with the specified container.

**Return** The set of file references associated with the container.

**Type** DataReferenceSet

**Example**

This example prints the location of files for two different components of a system.

```python
system1 = GetSystem(Name="Static1")
geometry1 = system1.GetContainer(ComponentName="Geometry")
for fileRef in geometry1.GetFiles():
    print fileRef.Location
>>> C:\Users\myUser\Projects\static1_files\dp0\SYS\DM\SYS.agdb
>>> E:\data\Models\pipe.x_t
model1 = system1.GetContainer(ComponentName="Model")
for fileRef in model1.GetFiles():
    print fileRef.Location
>>> C:\Users\myUser\Projects\static1_files\dp0\global\MECH\SYS.engd
>>> C:\Users\myUser\Projects\static1_files\dp0\global\MECH\SYS.mechdb
```

**Data Entities**

**FileReference**

The data entity that represents a file that is part of the project.

**Properties**

**Directory**

A string that contains the path to the directory containing the file.

**Type** string

**Read Only** Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

**Read Only** No
**Exists**

A value indicating whether the file exists.

**Type**  
bool

**ReadOnly**  
Yes

**FileName**

The name of the file, including the extension.

**Type**  
string

**ReadOnly**  
Yes

**LastModifiedTime**

The time at which this file was most recently modified.

**Type**  
DateTime

**ReadOnly**  
Yes

**Location**

The full path to the local file including the directory and file name.

**Type**  
string

**ReadOnly**  
No

**Size**

The size of the file in bytes.

**Type**  
long

**ReadOnly**  
Yes

**Methods**

**Copy**

Copies a file to a target directory.

**Return**  
A reference to the created file is returned in this argument.

**Type**  
DataReference

**Required Arguments**

**DestinationDirectoryPath**  
The full path to the destination directory.

**Type**  
string
Overwrite

Specifies whether to overwrite an existing files of the same name. Note:
A registered file may not be overwritten.
Type

bool

Example
The following example illustrates proper CopyFileByReference command invocation:

fileRef = GetRegisteredFileQuery(FilePath=r"path_to_file")
fileRefCopy = fileRef.Copy(DestinationDirectoryPath=r"path_to_copy_file", Overwrite=True, CopyReference

Delete
Deletes the specified file
Optional Arguments
BackUp

Specifies whether to back up the file before deletion. This optional argument's default
value is true.

DeleteIfShared

ErrorIfShared

Type

bool

Default Value

True

Specifies whether a registered file should be deleted even if it is still in use. This optional argument's default value is true. If this argument's value is false, an exception
will be thrown if a shared file is encountered. To avoid the exception, set ErrorIfShared
to false.
Type

bool

Default Value

False

Specifies whether to throw an exception when trying to delete a shared file if DeleteIfShared is set to false.
Type

bool

Example
The following example illustrates a deletion of a registered file via a file reference. The file will be deleted
if even if it is still in use. Note that the file will be backed up.
fileRef = GetRegisteredFile(FilePath=r"C:\Users\anyuser\path-to-file.extension")
fileRef.Delete(DeleteIfShared=True,
BackUp=True)

The next example illustrates a deletion of a registered file via a file reference without a forced deletion. If the file is shared, an error will not occur, and the file reference count will be decremented.
fileRef = GetRegisteredFile(FilePath=r"C:\Users\anyuser\path-to-file.extension")
fileRef.Delete(ErrorIfShared=False)

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information
of ANSYS, Inc. and its subsidiaries and affiliates.

529


Project

Move
Moves a file to a new directory.
Required Arguments
NewDirectoryPath

The full path to the destination directory.
Type

string

Optional Arguments
BackUp

Specifies whether to back up the file before the move. This optional argument's default value is
true.
Type

bool

Default Value

True

Example
The following example illustrates proper MoveFileByReference command invocation:
fileRef = GetRegisteredFileQuery(FilePath=r"path_to_file")
fileRefCopy = fileRef.Move(NewDirectoryPath=r"path_to_move_file", Backup=False)

Rename
Renames a file.
Required Arguments
NewName

The new file name, excluding the directory path. To change the directory, see MoveFileCommand.
Type

string

Optional Arguments
BackUp

Specifies whether to back up the file before renaming it. This optional argument's default value
is true.
Type

bool

Default Value

True

Example
The following example illustrates proper RenameFileByReference command invocation:
fileRef = GetRegisteredFileQuery(FilePath=r"path_to_file")
fileRefCopy = fileRef.Rename(NeName="new_file_name", BackUp=True)

Repair
Repairs a file that is referenced in the project but cannot be found on disk. If the file is expected to be
found under the project directory, then this command will copy the new file to the expected location

530

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information
of ANSYS, Inc. and its subsidiaries and affiliates.


rather than link to the new file in its current location. Use SetFilePathCommand to link to a file in a new location.

**Required Arguments**

**FilePath**
A full path to the file in its updated location.

  **Type** string

**RepositoryFileSource**

RepositoryFileSourceEntity defines repository file information including a server, workspace and filepath

**Properties**

**DisplayText**
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

  **Type** string
  **Read Only** No

**FileName**
The name of the file, including the extension.

  **Type** string
  **Read Only** Yes

**IsMissingInRepository**
A value indicating whether the file exists on repository server

  **Type** bool
  **Read Only** No

**LastDownloadRepositoryModificationTime**
Modified time stamp of file at the download time

  **Type** DateTime
  **Read Only** No

**LastDownloadTime**
Last downloaded time from repository server

  **Type** long
  **Read Only** No
LocalFileExists

A value indicating whether the local file exists.

Type bool
Read Only Yes

LocalLocation

String representation of local path to specify the local copy of repository file

Type string
Read Only No

Size

The size of the local file in bytes.

Type long
Read Only Yes

SourceFileInfo

Repository file info holder of

Type RepositoryFileInfo
Read Only No

Project Messages

This container holds current project messages.

Data Entities

StoredMessage

A data entity representing a message (error, warning, information, etc.) that is visible to the user.

Properties

Association

The data model entity to which the message applies (a component, for example).

Type string
Read Only Yes
**DateTimeStamp**

The publication time.

**Type**  
DateTime

**Read Only**  
Yes

**Details**

The detailed text string of the message.

**Type**  
string

**Read Only**  
Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

**MessageType**

The type of the message (e.g., Information, Warning, Error, etc.).

**Type**  
MessageType

**Read Only**  
Yes

**Summary**

The summary text string of the message.

**Type**  
string

**Read Only**  
Yes
System Coupling

System Coupling Setup
This container holds Set Up data for an instance of System Coupling.

Methods

CreateDataTransfer

Creates a data transfer.

\textbf{Return} \hspace{1cm} The new data transfer that was created.

\textbf{Type} \hspace{0.5cm} DataReference

Required Arguments

\textbf{Name} \hspace{0.5cm} The name for this data transfer.

\textbf{Type} \hspace{0.5cm} string

Optional Arguments

\textbf{SourceParticipant} \hspace{0.5cm} The source of this data transfer.

\textbf{Type} \hspace{0.5cm} DataReference

\textbf{SourceRegion} \hspace{0.5cm} The source Region for this data transfer.

\textbf{Type} \hspace{0.5cm} DataReference

\textbf{SourceVariable} \hspace{0.5cm} The source Variable for this data transfer.

\textbf{Type} \hspace{0.5cm} DataReference

\textbf{TargetParticipant} \hspace{0.5cm} The destination for this data transfer.

\textbf{Type} \hspace{0.5cm} DataReference

\textbf{TargetRegion} \hspace{0.5cm} The destination region for this data transfer.

\textbf{Type} \hspace{0.5cm} DataReference

\textbf{TargetVariable} \hspace{0.5cm} The destination variable for this data transfer.

\textbf{Type} \hspace{0.5cm} DataReference
Example
The following example demonstrates creation of a data transfer. It is assumed that user had created two participants (e.g. Transient Structural and Fluid Flow(Fluent) and connected to System Coupling System. System Coupling User's Guide in help documentation provides information on "Tutorial: Oscillating Plate with Two-Way Fluid-Structure Interaction". The same information can be referred to see how two participants can be setup and connect to System Coupling System.

```python
# Get System Coupling Setup
system1 = GetSystem(Name="SC")
setup1 = system1.GetContainer(ComponentName="Setup")
# Get source participant (e.g. Transient Structural), source region and source variable
participant1 = setup1.GetParticipant(Name="Solution")
region1 = participant1.GetRegion(Name="Fluid Solid Interface")
variable1 = region1.GetVariable(Name="Incremental Displacement")
# Get target participant (e.g. Fluid Flow(Fluent), target region and target variable
participant2 = setup1.GetParticipant(Name="Solution 1")
region2 = participant2.GetRegion(Name="wall_deforming")
variable2 = region2.GetVariable(Name="displacement")
# Create a data transfer
dataTransfer1 = setup1.CreateDataTransfer(
    Name="Data Transfer",
    SourceParticipant=participant1,
    SourceRegion=region1,
    SourceVariable=variable1,
    TargetParticipant=participant2,
    TargetRegion=region2,
    TargetVariable=variable2)
```

**ExportSCIFile**

Writes the system coupling input file at the specified path.

**Required Arguments**

Path  
Fully qualified path of the SCI File.

Type  
string

**GetAnalysisSettings**

Returns the analysis settings entity in a container.

**Return**

Data reference of the analysis settings in the container.

Type  
DataReference

**GetDataTransfer**

Returns the data transfer of a given name in a container.

**Return**

The data transfer that matches the specified name.

**Required Arguments**

Name  
The name or display name of the data transfer.
**GetDataTransfers**

Returns the collection of data transfers in a container. If no data transfers are in the container, the collection is empty.

**Return**

Collection of the data transfers in the container.

**Type** DataReferenceSet

**GetDebugOutputControls**

Returns the debug output controls from a system coupling setup container.

**Return**

Data reference of the debug output controls in the container.

**Type** DataReference

**GetExpertSettings**

Returns the expert settings entity from a system coupling setup container.

**Return**

Data reference of the expert settings in the container.

**Type** DataReference

**GetIntermediateRestartDataOutputControls**

Returns the intermediate result files output controls from a system coupling setup container. Renamed from ResultFiles to RestartData to avoid confusion for MAPDL users.

**Return**

Data reference of the intermediate result files output controls in the container.

**Type** DataReference

**GetParticipant**

Returns the participant with a given name from a system coupling setup container.

**Return**

The participant that matches the specified name.

**Type** DataReference

**Required Arguments**

**Name**

The internal name or display name of the participant.

**Type** string

**GetParticipants**

Returns the collection of participants in a container. If no participants are in the container, the collection is empty.
Return Collection of the participants in the container.

**Type**  DataReferenceSet

**GetSequenceControls**

Returns the sequence controls from a system coupling setup container.

Return Data reference of the sequence controls in the container.

**Type**  DataReference

**ReadRestartPoints**

Generates a list of restart points in analysis settings

**Data Entities**

**AnalysisSettings**

The entity to store the analysis settings for a coupling run.

**Properties**

**AnalysisType**

The coupled analysis type.

**Type**  CoupledAnalysisType

**Read Only**  No

**DisableSolutionUpdate**

This flag disables updates if restarts are not supported and solution data exists

**Type**  bool

**Read Only**  No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  string

**Read Only**  No

**DurationDefinedBy**

This property specifies how we should determine the end of the coupling run.

**Type**  DurationType
**EndTime**
The end time for the coupling run.

**Type** Quantity

**Read Only** No

**Initialization**
The initialization setting.

**Type** InitializationType

**Read Only** No

**MaximumIteration**
The maximum number of iterations per coupling step for the coupling run.

**Type** int

**Read Only** No

**MinimumIteration**
The minimum number of iterations per coupling step for the coupling run.

**Type** int

**Read Only** No

**NumberOfSteps**
The number of time steps for the coupling run.

**Type** int

**Read Only** No

**RestartStep**
The restart step for the coupling run.

**Type** int

**Read Only** No

**RestartTime**
The restart time for the coupling run.

**Type** Quantity
System Coupling

**StepSize**

The step size for the coupling run.

**Type**  
Quantity

**Read Only**  
No

**DataTransfer**

Entity to store a data transfer information.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

**IsSuppressed**

Suppression state of the entity.

**Type**  
bool

**Read Only**  
No

**SourceParticipant**

Participant system providing the data.

**Type**  
DataReference

**Read Only**  
No

**SourceRegion**

Participant region providing the data.

**Type**  
DataReference

**Read Only**  
No

**SourceVariable**

Variable provided by the source participant.

**Type**  
DataReference

**Read Only**  
No
**TargetParticipant**

Participant consuming the data.

**Type**  
DataReference

**Read Only**  
No

**TargetRegion**

Participant region consuming the data.

**Type**  
DataReference

**Read Only**  
No

**TargetVariable**

Variable consumed by the target participant.

**Type**  
DataReference

**Read Only**  
No

**TransferSettings**

Settings to specify how the data transfers are executed.

**Type**  
DataReference

**Read Only**  
No

**Methods**

**Delete**

Deletes a specified data transfer.

**Duplicate**

Duplicates the data transfer.

**Return**  
Data reference to the duplicate data transfer.

**Type**  
DataReference

**GetSettings**

Returns the settings for a specified data transfer.

**Return**  
The data transfer settings.

**Type**  
DataReference
**SetSuppression**

Suppresses or unsuppresses a data transfer

**Required Arguments**

**Suppressed**  The boolean value to specify if the item should be suppressed or unsuppressed

  - **Type**  bool

**DataTransferSettings**

Entity to store the settings for the data transfer control.

**Properties**

**ConvergenceTarget**

The target value used when evaluating convergence of the data transfer within a coupling iteration.

  - **Type**  double
  - **Read Only**  No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

  - **Type**  string
  - **Read Only**  No

**Ramping**

Setting that defines the ramping.

  - **Type**  RampingType
  - **Read Only**  No

**TransferAt**

Setting that defines when the transfers should happen.

  - **Type**  TransferAtType
  - **Read Only**  No

**UnderRelaxationFactor**

Convergence stability factor for highly non-linear couplings.

  - **Type**  double
  - **Read Only**  No
**DebugOutputControls**

Entity to store the debug level information for solution log.

**Properties**

**AnalysisInitialization**

This setting controls the level of output from the “Analysis Initialization” until the “Solution” synchronization point.

**Type** DebugLevel

**Read Only** No

**ConvergenceChecks**

This setting controls the level of output from the “Check Convergence” synchronization point until the next synchronization point, which may be either “Shutdown” or “Solution.”

**Type** DebugLevel

**Read Only** No

**DataTransfersLevel**

This setting provides the default level for the different kinds of debug output. If this entry is set and another specific entry (e.g., SourceMeshCoords) also exists, then the output level for the specific entry will override the level set here.

**Type** DebugLevel

**Read Only** No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

**Read Only** No

**GlobalLevel**

This setting provides the default level for the different sections of debug output. If this entry is set and another specific entry (e.g., Startup) also exists, then the output level for the specific entry will override the level set here.

**Type** DebugLevel

**Read Only** No
**ParticipantConnection**

This setting controls the level of output from the end of the setup validation until the “Analysis Initialization” synchronization point.

**Type** DebugLevel

**Read Only** No

**Shutdown**

This setting controls the level of output after the “Shutdown” synchronization point.

**Type** DebugLevel

**Read Only** No

**SolutionInitialization**

This setting controls the level of output during the setup of coupling steps and iterations. This output does not include information related to the data transfers.

**Type** DebugLevel

**Read Only** No

**SourceData**

This setting controls the level of output for the source data in all data transfers.

**Type** DebugLevel

**Read Only** No

**SourceMeshCoordinates**

This setting controls the level of output for mesh coordinates of the source region in all data transfers.

**Type** DebugLevel

**Read Only** No

**SourceMeshTopology**

This setting controls the level of output for mesh topology (elements and nodes) of the source region in all data transfers.

**Type** DebugLevel

**Read Only** No

**Startup**

This setting controls the level of output from the start of the coupling service until creation of the “Summary of SC Setup” banner in the SCL file.
**TargetData**
This setting controls the level of output for the target data in all data transfers.

**TargetMeshCoordinates**
This setting controls the level of output for mesh coordinates of the source region in all data transfers.

**TargetMeshTopology**
This setting controls the level of output for mesh topology (elements and nodes) of the source region in all data transfers.

**ExpertSettings**
The entity to store advanced options for data mapping.

**Properties**

**DisplayText**
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.
Methods

**AddProperty**

Adds a property to expert settings.

**Required Arguments**

- **Property**
  Property to be added.
  
  **Type**  string

**GetProperty**

Returns the property value for a specified property.

**Return**

The value of the property.

**Type**  string

**Required Arguments**

- **Property**
  The name of the property.
  
  **Type**  string

**RemoveProperty**

Removes an existing property from expert settings.

**Required Arguments**

- **Property**
  Property to be removed.
  
  **Type**  string

**SetProperty**

Sets the value of a property in expert settings.

**Required Arguments**

- **Property**
  The name of the property.
  
  **Type**  string

- **Value**
  The value of the property.
  
  **Type**  string

**IntermediateResultFilesOutputControls**

The entity to store the settings for result files creation during a coupling run.
Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string
Read Only: No

OutputFrequency

This property specifies the frequency at which the result files are generated.

Type: OutputFrequencyType
Read Only: No

StepInterval

The step interval at which the result files are generated.

Type: int
Read Only: No

SequenceControl

The entity to store solver sequence information for a coupling run.

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string
Read Only: No

Sequence

Dictionary containing sequence values for each participant.

Type: Dictionary<DataReference, int>
Read Only: No

Methods

GetSequence

Returns the sequence value for a participant.
Return

Sequence value for the given participant.

Type  int

Required Arguments

Participant

The participant for which to get the sequence.

Type  DataReference

SetSequence

This command will set the sequence number for the specified participant.

Required Arguments

Participant

The participant for which to set sequence value.

Type  DataReference

Value

Sequence value for the given participant.

Type  int

SystemCouplingCoSimulationParticipant

This is a base class for the entity which represents System Coupling Co-simulation Participant information.

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type  string

Read Only  No

InternalName

The internal name of the multiphysics participant (usually a solver) that is providing this data.

Type  string

Read Only  No

SystemCouplingParticipant

This is a base class for the entity which represents System Coupling Participant information. System Coupling Participant information comprises of System and solver-level information related to coupling. Note- Coupling participants are systems that will provide and/or consume data in a coupled analysis. Example systems in Workbench include: Analysis Systems – Steady-State Thermal, Transient Thermal, Static Structural, Transient Structural, Fluid Flow (Fluent) Component Systems – Fluent, External Data
Properties

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**InternalName**

The internal name of the multiphysics participant (usually a solver) that is providing this data

- **Type**: string
- **Read Only**: No

**SystemCouplingRegion**

This entity represents a System Coupling Region. A region is most often a point, line, surface or volume that is part (or all) of the geometry or topology of a coupling participant.

Properties

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**InternalName**

The name of the region as understood by the solver

- **Type**: string
- **Read Only**: No

**Topology**

The topology of the region

- **Type**: TopologyType
- **Read Only**: No

**SystemCouplingStaticDataParticipant**

This is a base class/entity for the entity which represents System Coupling Static Data Participant information.
System Coupling

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type string

Read Only No

InternalName

The internal name of the multiphysics participant (usually a solver) that is providing this data.

Type string

Read Only No

SystemCouplingSteadyCoSimulationParticipant

The entity represents System Coupling Steady Co-simulation Participant information e.g. information related to Static Structural Participant System for coupling purpose.

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type string

Read Only No

InternalName

The internal name of the multiphysics participant (usually a solver) that is providing this data.

Type string

Read Only No

SystemCouplingSteadyStaticDataParticipant

The entity represents System Coupling Steady Static Data Participant information e.g. information related to External Data Participant System for coupling purposes.

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.
**Type**  
string

**Read Only**  
No

**InternalName**

The internal name of the multiphysics participant (usually a solver) that is providing this data

**Type**  
string

**Read Only**  
No

**SystemCouplingTransientCoSimulationParticipant**

The entity represents System Coupling Transient Co-simulation Participant information e.g. information related to Transient Structural Participant System for coupling purposes.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

**InternalName**

The internal name of the multiphysics participant (usually a solver) that is providing this data

**Type**  
string

**Read Only**  
No

**SystemCouplingTransientStaticDataParticipant**

The entity represents System Coupling Transient Static Data Participant information.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

**InternalName**

The internal name of the multiphysics participant (usually a solver) that is providing this data
This entity represents a System Coupling Variable. A variable is a physical quantity such as force, length, or temperature that can be transferred between regions of participant systems.

**Properties**

**DataType**

The tensor type of the variable

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**Exposure**

The type of transfer that will be done with this variable

<table>
<thead>
<tr>
<th>Type</th>
<th>VariableExposure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**InternalName**

The name of the variable

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**PhysicalType**

The quantity type of the variable

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>
System Coupling Solution

This container holds Solution data for an instance of System Coupling.

Methods

**CreateConvergenceChart**

Creates a new convergence chart.

**Return**

The data reference to the new convergence chart.

**Type**

DataReference

**Required Arguments**

**Name**

The name for this Convergence Chart.

**Type**

string

**CreateSolutionInformation**

Creates a new solution information entity and adds it to the specified system coupling solution container.

**Return**

The new solution information that was created.

**Type**

DataReference

**Required Arguments**

**Name**

The name for this Solution Information.

**Type**

string

**SolutionInformationFilePath**

The path of the file to read solution information.

**Type**

string

**GetAllSolutionInformation**

Returns the collection of solution information entities in a container. If no solution information entities are in the container, the collection is empty.

**Return**

Collection of the solution informations in the container.

**Type**

DataReferenceSet

**GetChartVariableNames**

Returns a dictionary of fully qualified chart variable names and display names.

During System Coupling Solution cell Update or after Solution cell Update, these names can be used to create chart variable using "CreateVariable" data entity method of "ConvergenceChart".
Data transfer chart variable

Qualified Name Format - "Target Participant Internal Name":"Data Transfer Internal Name":"Variable Name": "Operator Name"
Qualified Name Example- "Solution 1:Data Transfer 1:Change:Maximum"
Display Name Example- "Fluid Flow (FLUENT):Data Transfer 1:Change:Maximum"

Solver chart variable

Qualified Name Format - "Participant Internal Name":"Variable Name"
Qualified Name Example- "Solution 1:Continuity Convergence"
Display Name Example- "Fluid Flow (FLUENT):Continuity Convergence"

Return A dictionary of fully qualified chart variable names and display names

Please see summary documentation of this (GetChartVariableNames) query on details of format and example of fully qualified name.

Type Dictionary<string, string>

GetConvergenceChart

Returns the convergence chart of a given name in a container.

Return The convergence chart that matches the specified name.

Type DataReference

Required Arguments

Name The name or display name of the convergence chart.

Type string

GetConvergenceCharts

Returns the collection of convergence charts in a container. If no convergence charts are in the container, the collection is empty.

Return Collection of the convergence charts in the container.

Type DataReferenceSet

GetSolutionComponentProperties

Returns the solution component properties for a system coupling solution container.

Return Data reference to the properties objects.

Type DataReference

GetSolutionInformation

Returns the solution information of a given name in a container.
**Return**

The solution information that matches the specified name.

**Type**  
DataReference

**Required Arguments**

**Name**  
The name or display name of the solution information.

**Type**  
string

**Data Entities**

**AxisContinuous**

A chart axis that spans a set of continuous values. An example is an axis of an XY plot

**Properties**

**AutomaticRange**

The property to define whether or not automatic scaling should be applied to the axis, or whether the RangeMin and RangeMax should be used.

**Type**  
bool

**Read Only**  
No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

**QuantityName**

The name of the quantity associated with axis data.

E.g. Coupling Iteration, Coupling Step, Coupling Time

**Type**  
string

**Read Only**  
No

**RangeMaximum**

The maximum range of the values in this axis.

**Type**  
double

**Read Only**  
No
**RangeMinimum**

The minimum range of the values in this axis.

**Type**  
double

**Read Only**  
No

**Scale**

The scale of the axis. Scale can be defined as Linear/CommonLog (Log base 10)/Natural Log.

**Type**  
Scale

**Read Only**  
No

**Title**

The title of the axis.

**Type**  
string

**Read Only**  
No

**ChartVariable**

Entity representing a variable in Convergence Chart

**Properties**

**Color**

The line color of this chart variable in a plot.

This property is valid only for the chart displayed in Scene View.

**Type**  
Color

**Read Only**  
No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

**LineWidth**

The width of the line drawn for this chart variable in pixels.

This property is valid only for the chart displayed in Scene View.

**Type**  
float
**QualifiedName**

The variable quantity to display.

- **Type**: string
- **Read Only**: No

**RefinementLevel**

Refinement level for the data to be plotted.

- **Type**: string
- **Read Only**: No

**SymbolSize**

The size of a symbol in pixels when a symbol is drawn for this variable. The rendered symbol size may be slightly smaller or larger than expected if symbol does not correctly fit into the specified number of pixels.

This property is valid only for the chart displayed in Scene View.

- **Type**: uint
- **Read Only**: No

**Methods**

**Delete**

Deletes a specified chart variable.

**ConvergenceChart**

Entity to store a convergence chart information.

**Properties**

**AxisX**

Associated X Axis

- **Type**: DataReference
- **Read Only**: No

**AxisY**

Associated Y Axis

- **Type**: DataReference
Read Only No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type string

Read Only No

**Variables**

Collection of variables to be plotted.

Type DataReferenceSet

Read Only No

**XAxis**

X Axis Quantity Name

Type string

Read Only No

**Methods**

**CreateVariable**

Creates a chart variable based on specified qualified name and adds it to the specified convergence chart.

During System Coupling Solution cell Update or after Solution cell Update, user can create convergence chart and add chart variables. A chart variable is based on specified fully qualified name. GetChartVariableNames query returns a dictionary of fully qualified chart variable names and display names. This can be used to create charts.

Data transfer chart variable format

Qualified Name Format - "Target Participant Internal Name":"Data Transfer Internal Name":"Variable Name": "Operator Name"
Qualified Name Example- "Solution 1:Data Transfer 1:Change:Maximum"
Display Name Example- "Fluid Flow (FLUENT):Data Transfer 1:Change:Maximum"

Solver chart variable format

Qualified Name Format - "Participant Internal Name":"Variable Name"
Qualified Name Example- "Solution 1:Continuity Convergence"
Display Name Example- "Fluid Flow (FLUENT):Continuity Convergence"

**Return** The new created chart variable.

Type DataReference
Required Arguments

**QualifiedName**

The fully qualified name for this chart variable.

Please see summary documentation of this (CreateVariable) data entity method on details of format and example of fully qualified name.

**Type** string

Optional Arguments

**DisplayName**

The display name for this chart variable.

**Type** string

Example

The following example demonstrates creation of chart variables. It is assumed that user has setup participants (e.g. Transient Structural and Fluid Flow(FLUENT) and System Coupling system, and also solved coupled analysis.

```plaintext
# Get System Coupling Solution
system1 = GetSystem(Name="SC")
solution1 = system1.GetContainer(ComponentName="Solution")
# Create Convergence Chart
ConvergenceChart1 = solution1.CreateConvergenceChart(Name="Chart")
# Create a Data Transfer chart variable
ChartVariable1 = ConvergenceChart1.CreateVariable(QualifiedName="Solution 1:Data Transfer 1:Change:Maximum",DisplayName="Fluid Flow (FLUENT):Data Transfer 1:Change:Maximum")
# Create a solver chart variable
ChartVariable2 = ConvergenceChart1.CreateVariable(QualifiedName="Solution 1:Continuity Convergence",DisplayN
```

**Delete**

Delete's a specified convergence chart.

**GetAxis**

Returns the axis for a specified convergence chart

**Return** The axis

**Type** DataReference

Required Arguments

**Name** Name of the Axis

**Type** string

**GetChartVariable**

Returns the chart variable of a given name from a convergence chart

**Return** The chart variable that matches the specified name.

**Type** DataReference
Required Arguments

**QualifiedName**
The qualified name of the chart variable.

**Type**  string

*GetChartVariables*

Returns the collection of chart variables for a given convergence chart

**Return**
A collection of the variables in the chart.

**Type**  DataReferenceSet

*SolutionComponentProperties*

The entity to store additional command line options which are passed to the coupling service on update.

**Properties**

*CommandLineOptions*

Additional command line options which are passed to the coupling service on update.

**Type**  string

**Read Only**  No

*DisplayText*

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  string

**Read Only**  No

*SolutionInformation*

Entity to store the solution information provided by the coupled participants.

**Properties**

*DisplayText*

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  string

**Read Only**  No

*FilePath*

The path of the solution information file.
Type       string
Read Only  No

Methods

Delete

Deletes a specified solution information entity.
TurboSystems

Turbo Geometry
This container holds Geometry data for an instance of BladeGen.

Methods

CreateBladeMesh
Creates a new Mesh system and automatically meshes the fluid zone for the blade geometry from the Blade Design component.

CreateGeometry
This command class creates a new BladeEditor model. An up-to-date Geometry cell appears on the project schematic containing the new model.

Edit
Opens the BladeGen editor to allow modification of Blade Design data.

This command will open the editor only if one is not already open on this component. If this component's editor is already open, then it will be raised to the front.

Exit
Exits the BladeGen editor.

Any changes made in this editor will be retained on exit. These changes are made permanent by a Project Save, and will be discarded in the event of closing the project without saving. If no editor is open on the component in question, this command will have no effect.

Optional Arguments

ExitApp
This parameter is deprecated and will be ignored.

Type bool
Default Value True

GetTurboGeometryProperties
Returns the Data Entity which contains user settings and properties for the Blade Design container.

Return A reference to the requested data entity.
**Import**

Imports blade geometry data into the BladeGen editor from an existing BladeGen file.

**Required Arguments**

**FilePath**  
Command argument containing the file name to be opened.

**Type**  
`string`

**Example**

```plaintext
template1 = GetTemplate(TemplateName="BladeGen")
system1 = template1.CreateSystem()
bladeDesign1 = system1.GetContainer(ComponentName="Blade Design")
bladeDesign1.Import(FilePath="myfilepath/pump.bgd")
```

**Data Entities**

**TurboGeometryProperties**

This data entity provides access to the import properties that are used to determine how the blade geometry is handled when it is transferred to a downstream Geometry cell.

**Properties**

**BladeExt**

Import Option that specifies the blade surface extension length (as a percentage of the average hub to shroud distance) when the Blade Design data is transferred to a downstream Geometry. These surfaces are extended and then trimmed to the MasterProfile sketch to ensure that the blade solid correctly matches the hub and shroud contours.

**Type**  
`double`

**Read Only**  
No

**BladeLoftOption**

Import Option that specifies how to loft the blade surfaces when the Blade Design data is transferred to a downstream Geometry.

Available options:

- **Streamwise**  
  Loft the blade surfaces in the streamwise direction through curves that run from hub to shroud.

- **Spanwise**  
  Loft the blade surfaces in the spanwise direction through the blade profile curves.

**Type**  
`BladeLoftType`

**Read Only**  
No
**CreateAllBlades**

Import Option that specifies whether to create one or all blades when the Blade Design data is transferred to a downstream Geometry.

Available options:

- **True**
  - BladeEditor will create all the blades using the number of blades specified in the BladeGen model.

- **False**
  - Only one blade will be created.

**Type**

bool

**Read Only**

No

**CreateFluidZone**

Import Option that specifies whether to create the fluid zone body when the Blade Design data is transferred to a downstream Geometry. The resulting Enclosure can be used for a CFD analysis of the blade passage.

Available options:

- **True**
  - Create the fluid zone Enclosure.

- **False**
  - Don't create the fluid zone Enclosure.

**Type**

bool

**Read Only**

No

**CreateHub**

Import Option that specifies whether a hub body will be created when the Blade Design data is transferred to a downstream Geometry.

Available options:

- **True**
  - BladeEditor will create a HubProfile sketch for the non-flow path hub geometry, and will create a revolved body feature called HubBody.

- **False**
  - BladeEditor will not create the hub body.

**Type**

bool

**Read Only**

No

**CreateNamedSelections**

Import Option that specifies whether to create the Named Selections for the fluid zone when the Blade Design data is transferred to a downstream Geometry. If this property is selected, then BladeEditor will create Named Selections (regions) for the typical faces of the blade passage, i.e., Blade, Hub, Shroud, Inflow, Outflow, PeriodicA and PeriodicB. These Named Selections can be used as selection groups in
other ANSYS Workbench applications. Note that this property is available only if Create Fluid Zone is selected.

Available options:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Create the Named Selections.</td>
</tr>
<tr>
<td>False</td>
<td>Don't create the Named Selections.</td>
</tr>
</tbody>
</table>

**Type**  
bool

**Read Only**  
No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

**LayerNumber**

Import Option that specifies the integer value of the Layer Number to use for the shroud clearance when the Blade Design data is transferred to a downstream Geometry. This property only applies when the Shroud Clearance property is set to Relative Layer or Absolute Layer.

**Type**  
int

**Read Only**  
No

**MachineType**

Specification of the machine type, used by the downstream VistaTF setup

Available options:

- Pump
- AxialCompressor
- CentrifugalCompressor
- Fan
- AxialTurbine
- RadialTurbine
- HydraulicTurbine
- Other
- Unknown

**Type**  
MachineType

**Read Only**  
No

**MergeBladeTopology**

Import Option that specifies whether to merge the blade faces when the Blade Design data is transferred to a downstream Geometry. If not merged, there will be four faces corresponding to the leading edge,
trailing edge, pressure and suction surfaces of the blade. If merged, blade faces that are tangent to one another will be merged into a single face.

Available options:

- True: Merge the blade faces where they are tangent to one another.
- False: Don't merge the blade faces.

**ModelUnits**

The length scale units the BladeGen model was created in. Used when transferring the BladeGen model to VistaTF.

Available options:

- m
- cm
- mm
- inches
- ft

**PeriodicSurfExt**

Import Option that defines the periodic surface extension length (as a percentage of the average hub to shroud distance) when the Blade Design data is transferred to a downstream Geometry.

**PeriodicSurfOption**

Import Option that specifies the style of the periodic interface surfaces for the fluid zone when the Blade Design data is transferred to a downstream Geometry. Note that this property is available only if Create Fluid Zone is selected.

Available options:

- OnePiece: The periodic surface is created as a single surface.
- ThreePieces: The periodic surface is created in three connected pieces: one upstream of the blade, one within the passage, and one downstream of the blade. This style can better accommodate highly curved or twisted blades, and is similar
to the ANSYS TurboGrid style of periodic surface.

**Type** PeriodicSurfType

**Read Only** No

**ShroudClearance**

Import Option that specifies whether to include the shroud clearance when the Blade Design data is transferred to a downstream Geometry.

Available options:

- **None**
  - No shroud clearance is created.
- **RelativeLayer**
  - The selected Layer Number is relative to the shroud layer, e.g., 1 implies the first layer closest to the shroud layer, 2 implies the second closest layer to the shroud, etc.
- **AbsoluteLayer**
  - The selected layer index counts up from the hub layer, which is zero.

**Type** ClearanceType

**Read Only** No

**SpanwiseCount**

The number of spanwise gridlines used in the downstream VistaTF calculation. Default = 4

**Type** int

**Read Only** No

**StreamwiseCount**

The number of streamwise gridlines used in the downstream VistaTF calculation. Default = 20

**Type** int

**Read Only** No

---

**Turbo Mesh**

This container holds Mesh data for an instance of TurboGrid.

**Methods**

**Edit**

Opens the TurboGrid editor to allow modification of Turbo Mesh data.
Optional Arguments

Interactive
Run the editor in interactive mode if True, or in no GUI mode if False.

If not specified, the editor runs in interactive mode.

Type: bool
Default Value: True

TopologySuspended
If True, open the editor with the topology in a suspended state. Otherwise, open the editor with the suspended state of topology the same as when the editor was last closed.

If not specified, it defaults to false.

Type: bool
Default Value: False

Exit
Exits the editor.

Any changes made in this editor will be retained on exit. These changes are made permanent by a Project Save, and will be discarded in the event of closing the project without saving.

If no editor is open on the component in question, this command will have no effect.

GetTurboMeshProperties
Returns the Data Entity which contains user settings and properties for the Turbo Mesh container.

Return
A reference to the requested data entity.

Type: DataReference

SendCommand
Sends commands to the editor for this component using CFX Command Language (CCL) syntax. If the editor for this component is not open, it will be launched before the commands are sent and subsequently closed. In this mode, component data is loaded and saved as if calling Edit(Interactive=False) and Exit around the SendCommand invocation.

The instructions must be CFX Command Language session commands that are valid for the editor in question.

Required Arguments

Command
Valid CFX Command Language (CCL) commands

Type: string
Data Entities

TurboMeshProperties

This data entity provides access to the properties that are used to determine which blade geometry to mesh and how to handle the inlet and outlet positions.

Properties

AvailableFlowpaths

Displays a list of the available flowpaths and bladerows. Use this information as a guide when specifying the Flowpath and Bladerow properties (described below). Use the Refresh command in the context menu to update the list after linking.

Type string
Read Only Yes

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type string
Read Only No

DownstreamBladerowNumber

Specifies the bladerow number for the bladerow that is immediately downstream of the current bladerow. This property is available only when Outlet Position Method is set to Adjacent Blade.

Type int
Read Only No

InletBlock

If this is checked then an inlet block will be generated, if possible, when the mesh is created.

Type bool
Read Only No

InletPositionOption

Specifies how the inlet points are positioned in TurboGrid. This property is only available when multiple bladerows in the same flowpath have been exported from BladeEditor.

Available options:

Manual The user will specify the inlet points in TurboGrid.
TurboGrid will calculate the inlet points to be halfway between the selected upstream bladerow and the current bladerow.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AdjacentBlade</strong></td>
<td>TurboGrid will calculate the inlet points to be halfway between the selected upstream bladerow and the current bladerow.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>OpeningPositionMethod</td>
</tr>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>

**MaximumFaceAngle**

This is the maximum face angle in the mesh if the mesh has been generated.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MaximumFaceAngle</strong></td>
<td>This is the maximum face angle in the mesh if the mesh has been generated.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Quantity</td>
</tr>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>

**MeshNamePrefix**

If specified, this string will be prepended to all region names in the mesh when transferred to a CFX system. This option is only available when Beta features are enabled.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MeshNamePrefix</strong></td>
<td>If specified, this string will be prepended to all region names in the mesh when transferred to a CFX system. This option is only available when Beta features are enabled.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>string</td>
</tr>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>

**MinimumFaceAngle**

This is the minimum face angle in the mesh if the mesh has been generated.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MinimumFaceAngle</strong></td>
<td>This is the minimum face angle in the mesh if the mesh has been generated.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Quantity</td>
</tr>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>

**OutletBlock**

If this is checked then an outlet block will be generated, if possible, when the mesh is created.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OutletBlock</strong></td>
<td>If this is checked then an outlet block will be generated, if possible, when the mesh is created.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Read Only</strong></td>
<td>No</td>
</tr>
</tbody>
</table>

**OutletPositionOption**

Specifies how the outlet points are positioned in TurboGrid. This property is only available when multiple bladerows in the same flowpath have been exported from BladeEditor.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OutletPositionOption</strong></td>
<td>Specifies how the outlet points are positioned in TurboGrid. This property is only available when multiple bladerows in the same flowpath have been exported from BladeEditor.</td>
</tr>
<tr>
<td><strong>Available options:</strong></td>
<td></td>
</tr>
<tr>
<td>Manual</td>
<td>The user will specify the outlet points in TurboGrid.</td>
</tr>
<tr>
<td>AdjacentBlade</td>
<td>TurboGrid will calculate the outlet points to be halfway between the selected downstream bladerow and the current bladerow.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>OpeningPositionMethod</td>
</tr>
</tbody>
</table>
**SelectedBladerowNum**

Specifies which bladerow (within the specified Flowpath feature) is to be loaded in ANSYS TurboGrid. The bladerows are number sequentially, starting from 1 for native Blade features.

*Type* `int`  
*Read Only* No

**SelectedFlowpathName**

Specifies which Flowpath feature in BladeEditor contains the bladerow that is to be loaded in ANSYS TurboGrid. On initial refresh, it will default to the first flowpath available.

*Type* `string`  
*Read Only* No

**UpstreamBladerowNumber**

Specifies the bladerow number for the bladerow that is immediately upstream of the current bladerow. This property is available only when Inlet Position Method is set to Adjacent Blade.

*Type* `int`  
*Read Only* No

---

**Vista AFD Analysis**

This container holds Analysis data for an instance of Vista AFD.

**Methods**

**CreateBladeDesign**

This command class creates a new BladeGen model. An up-to-date BladeGen cell appears on the project schematic containing the new model.

**Optional Arguments**

**Beta**  
Option to use the beta BladeGen template

*Type* `bool`  
*Default Value* False

**CreateGeometry**

This command class creates a new BladeEditor model. An up-to-date Geometry cell appears on the project schematic containing the new model.
**Edit**

This command class launches the Vista popup GUI.

**GetVistaAFDAnalysisProperties**

This query takes a container reference and returns the Data Entity which contains user settings and properties for the VistaAFD Analysis container.

Return

Reference to the requested Data Entity

Type

DataReference

**Data Entities**

**VistaAFDAnalysis**

Analysis represents a VistaAFD throughflow calculation used to analyse the design performance

**Properties**

**Alpha1**

User input, IGV exit angle

Type

Quantity

Read Only

No

**Alpha3**

User input, OGV exit angle

Type

Quantity

Read Only

No

**AlphaOGVHub**

OGV hub gas exit angle

Type

Quantity

Read Only

No

**AlphaOGVMean**

OGV mean gas exit angle

Type

Quantity

Read Only

No
**AlphaRotHub**

Rotor hub gas exit angle

- **Type**: Quantity
- **Read Only**: No

**AlphaRotMean**

Rotor mean gas exit angle

- **Type**: Quantity
- **Read Only**: No

**Blade**

Blade number to export to Bladegen (0=IGV, 1=Rotor, 2=OGV)

- **Type**: int
- **Read Only**: No

**BladeBetaExit**

Blade exit angles

- **Type**: List<List<float>>
- **Read Only**: No

**BladeBetaInlet**

Blade inlet angles

- **Type**: List<List<float>>
- **Read Only**: No

**BladeOption**

Blade option for export to BladeGen

- **Type**: BladeType
- **Read Only**: No

**BMunits**

BladeGen/BladeEditor units

- **Type**: string
- **Read Only**: No
**BMunitsOption**

BladeGen/BladeEditor units option

**Type**  BMunitsType

**Read Only**  No

**DeHallerOGVHub**

OGV hub DeHaller number

**Type**  float

**Read Only**  No

**DeHallerOGVMean**

OGV mean DeHaller number

**Type**  float

**Read Only**  No

**DeHallerRotHub**

Rotor hub DeHaller number

**Type**  float

**Read Only**  No

**DeHallerRotMean**

Rotor mean DeHaller number

**Type**  float

**Read Only**  No

**DevIGVHub**

IGV hub deviation

**Type**  Quantity

**Read Only**  No

**DevIGVMean**

IGV mean deviation

**Type**  Quantity

**Read Only**  No
DevOGVHub
OGV hub deviation
Type Quantity
Read Only No

DevOGVMean
OGV mean deviation
Type Quantity
Read Only No

DevRotHub
Rotor hub deviation
Type Quantity
Read Only No

DevRotMean
Rotor mean deviation
Type Quantity
Read Only No

DfOGVHub
OGV hub diffusion factor
Type float
Read Only No

DfOGVMean
OGV mean diffusion factor
Type float
Read Only No

DfRotHub
Rotor hub diffusion factor
Type float
Read Only No
**DfRotMean**

Rotor mean diffusion factor

**Type** float

**Read Only** No

**Diameter**

User input, outer diameter

**Type** Quantity

**Read Only** No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type** string

**Read Only** No

**Eta**

Aerodynamic efficiency

**Type** float

**Read Only** No

**EtaInput**

User input, efficiency estimate

**Type** float

**Read Only** No

**EtaTS**

System efficiency (t-s)

**Type** float

**Read Only** No

**EtaTSPipe**

Downstream system efficiency (t-s)

**Type** float

**Read Only** No
*EtaTT*

System efficiency (t-t)

**Type** float

**Read Only** No

*HeadRise*

User input, total head rise

**Type** Quantity

**Read Only** No

*HtrIn*

User input, hub/tip rotor inlet

**Type** float

**Read Only** No

*HtrOut*

User input, hub/tip rotor outlet

**Type** float

**Read Only** No

*HubLoadParam*

User input, hub loading parameter

**Type** float

**Read Only** No

*HubVelFactor*

User input, hub velocity deficit factor

**Type** float

**Read Only** No

*HubX*

Hub annulus X-coords

**Type** List<Quantity>

**Read Only** No
**HubY**

Hub annulus Y-coords

**Type** List<Quantity>

**Read Only** No

**IGV**

User input, IGV option

**Type** bool

**Read Only** No

**IGVhubThickX**

IGV hub thickness X-coord

**Type** List<float>

**Read Only** No

**IGVhubThickY**

IGV hub thickness Y-coord

**Type** List<Quantity>

**Read Only** No

**IGVleadingX**

IGV leading edge X-coords

**Type** List<Quantity>

**Read Only** No

**IGVleadingY**

IGV leading edge Y-coords

**Type** List<Quantity>

**Read Only** No

**IGVshrThickX**

IGV shroud thickness X-coord

**Type** List<float>

**Read Only** No
**IGVshrThickY**

IGV shroud thickness Y-coord

Type: List<Quantity>

Read Only: No

**IGVthetaLE**

IGV leading edge theta

Type: List<float>

Read Only: No

**IGVtrailingX**

IGV trailing edge X-coords

Type: List<Quantity>

Read Only: No

**IGVtrailingY**

IGV trailing edge Y-coords

Type: List<Quantity>

Read Only: No

**ImperialUnits**

User input, Imperial units option

Type: bool

Read Only: No

**InnerIter**

User input, number of inner loop design calculation iterations

Type: int

Read Only: No

**Layer1**

Intermediate spanwise layer1 for Export

Type: bool

Read Only: No
Layer2
Intermediate spanwise layer 2 for Export
Type bool
Read Only No

Layer3
Intermediate spanwise layer 3 for Export
Type bool
Read Only No

LoadHub
Rotor hub loading
Type float
Read Only No

LoadMean
Rotor mean loading
Type float
Read Only No

MassFlow
User input, mass flow rate
Type Quantity
Read Only No

MaxLoadHub
Rotor maximum hub loading
Type float
Read Only No

MaxLoadMean
Rotor maximum mean loading
Type float
Read Only No
MixLoss
User input, downstream mixing losses

Type float
Read Only No

NMain
Number of blades in each row

Type List<int>
Read Only No

OGV
User input, OGV option

Type bool
Read Only No

OGVhubThickX
OGV hub thickness X-coord

Type List<float>
Read Only No

OGVhubThickY
OGV hub thickness Y-coord

Type List<Quantity>
Read Only No

OGVleadingX
OGV leading edge X-coords

Type List<Quantity>
Read Only No

OGVleadingY
OGV leading edge Y-coords

Type List<Quantity>
Read Only No
**OGVshrThickX**
OGV shroud thickness X-coord

**Type** List<float>

**Read Only** No

**OGVshrThickY**
OGV shroud thickness Y-coord

**Type** List<Quantity>

**Read Only** No

**OGVthetaLE**
OGV leading edge theta

**Type** List<float>

**Read Only** No

**OGVtrailingX**
OGV trailing edge X-coords

**Type** List<Quantity>

**Read Only** No

**OGVtrailingY**
OGV trailing edge Y-coords

**Type** List<Quantity>

**Read Only** No

**Omega**
User input, rotational speed

**Type** Quantity

**Read Only** No

**OuterIter**
User input, number of outer loop design calculation iterations

**Type** int

**Read Only** No
**Pdyn**
Outlet dynamic pressure

**Type** Quantity
**Read Only** No

**PdynPipe**
Downstream dynamic pressure

**Type** Quantity
**Read Only** No

**PhiHub**
Rotor hub flow coefficient

**Type** float
**Read Only** No

**PhiMean**
Rotor mean flow coefficient

**Type** float
**Read Only** No

**Power**
Power

**Type** Quantity
**Read Only** No

**RatioIGV**
User input, IGV aspect ratio

**Type** float
**Read Only** No

**RatioOGV**
User input, OGV aspect ratio

**Type** float
**Read Only** No
**RatioRotor**
User input, Rotor aspect ratio

**Type** float
**Read Only** No

**RotorHubThickX**
Rotor hub thickness X-coord

**Type** List<float>
**Read Only** No

**RotorHubThickY**
Rotor hub thickness Y-coord

**Type** List<Quantity>
**Read Only** No

**RotorLeadingX**
Rotor leading edge X-coords

**Type** List<Quantity>
**Read Only** No

**RotorLeadingY**
Rotor leading edge Y-coords

**Type** List<Quantity>
**Read Only** No

**RotorShrThickX**
Rotor shroud thickness X-coord

**Type** List<float>
**Read Only** No

**RotorShrThickY**
Rotor shroud thickness Y-coord

**Type** List<Quantity>
**Read Only** No
**RotorThetaLE**

Rotor leading edge theta

**Type**  List<float>

**Read Only**  No

**RotorTrailingX**

Rotor trailing edge X-coords

**Type**  List<Quantity>

**Read Only**  No

**RotorTrailingY**

Rotor trailing edge Y-coords

**Type**  List<Quantity>

**Read Only**  No

**Sc90MaxIter**

User input, maximum number of solver iterations

**Type**  int

**Read Only**  No

**Sc90Relax**

User input, solver relaxation factor

**Type**  float

**Read Only**  No

**Sc90Tol**

User input, solver tolerance

**Type**  float

**Read Only**  No

**ShrX**

Shroud annulus X-coords

**Type**  List<Quantity>

**Read Only**  No
**ShrY**
Shroud annulus Y-coords

**Type** List<Quantity>

**Read Only** No

**Slunits**
User input, SI units option

**Type** bool

**Read Only** No

**Span**
Spanwise fractions

**Type** List<float>

**Read Only** No

**StagPressure**
User input, inlet stagnation pressure

**Type** Quantity

**Read Only** No

**StagTemp**
User input, inlet stagnation temperature

**Type** Quantity

**Read Only** No

**Torque**
Torque

**Type** Quantity

**Read Only** No

**TrimIGV**
User input, IGV profile trim

**Type** float

**Read Only** No
**TrimOGV**

User input, OGV profile trim

Type float

Read Only No

**TrimRotor**

User input, Rotor profile trim

Type float

Read Only No

**VanesIGV**

User input, IGV number of vanes

Type int

Read Only No

**VanesOGV**

User input, OGV number of vanes

Type int

Read Only No

**VanesRotor**

User input, Rotor number of vanes

Type int

Read Only No

**VistaAFDTitle**

Editor Title

Type string

Read Only No
Vista AFD Design
This container holds Design data for an instance of Vista AFD.

Methods

CreateBladeDesign
This command class creates a new BladeGen model. An up-to-date BladeGen cell appears on the project schematic containing the new model.

Optional Arguments

Beta  Option to use the beta BladeGen template

  Type   bool
  Default Value   False

CreateGeometry
This command class creates a new BladeEditor model. An up-to-date Geometry cell appears on the project schematic containing the new model.

Edit
This command class launches the Vista popup GUI.

GetVistaAFDDesignProperties
This query takes a container reference and returns the Data Entity which contains user settings and properties for the VistaAFD Design container.

  Return   Reference to the requested Data Entity
  Type   DataReference

Data Entities

VistaAFDDesign
Design represents a VistaAFD throughflow calculation used to define the blade profiles

Properties

Alpha1
User input, IGV exit angle

  Type   Quantity
  Read Only   No
**Alpha3**
User input, OGV exit angle

**Type** Quantity

**Read Only** No

**AlphaOGVHub**
OGV hub gas exit angle

**Type** Quantity

**Read Only** No

**AlphaOGVMean**
OGV mean gas exit angle

**Type** Quantity

**Read Only** No

**AlphaRotHub**
Rotor hub gas exit angle

**Type** Quantity

**Read Only** No

**AlphaRotMean**
Rotor mean gas exit angle

**Type** Quantity

**Read Only** No

**Blade**
Blade number to export to Bladegen (0=IGV, 1=Rotor, 2=OGV)

**Type** int

**Read Only** No

**BladeBetaExit**
Blade exit angles

**Type** List<List<float>>

**Read Only** No
**BladeBetaInlet**

Blade inlet angles

**Type** List<List<float>>

**Read Only** No

**BladeOption**

Blade option for export to BladeGen

**Type** BladeType

**Read Only** No

**BMunits**

BladeGen/BladeEditor units

**Type** string

**Read Only** No

**BMunitsOption**

BladeGen/BladeEditor units option

**Type** BMunitsType

**Read Only** No

**DeHallerOGVHub**

OGV hub DeHaller number

**Type** float

**Read Only** No

**DeHallerOGVMean**

OGV mean DeHaller number

**Type** float

**Read Only** No

**DeHallerRotHub**

Rotor hub DeHaller number

**Type** float

**Read Only** No
**DeHallerRotMean**

Rotor mean DeHaller number

Type: float
Read Only: No

**DevIGVHub**

IGV hub deviation

Type: Quantity
Read Only: No

**DevIGVMean**

IGV mean deviation

Type: Quantity
Read Only: No

**DevOGVHub**

OGV hub deviation

Type: Quantity
Read Only: No

**DevOGVMean**

OGV mean deviation

Type: Quantity
Read Only: No

**DevRotHub**

Rotor hub deviation

Type: Quantity
Read Only: No

**DevRotMean**

Rotor mean deviation

Type: Quantity
Read Only: No
**DfOGVHub**

OGV hub diffusion factor

**Type**  
float

**Read Only**  
No

**DfOGVMean**

OGV mean diffusion factor

**Type**  
float

**Read Only**  
No

**DfRotHub**

Rotor hub diffusion factor

**Type**  
float

**Read Only**  
No

**DfRotMean**

Rotor mean diffusion factor

**Type**  
float

**Read Only**  
No

**Diameter**

User input, outer diameter

**Type**  
Quantity

**Read Only**  
No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

**Eta**

Aerodynamic efficiency

**Type**  
float

**Read Only**  
No
**EtaInput**
User input, efficiency estimate

Type float
Read Only No

**EtaTS**
System efficiency (t-s)

Type float
Read Only No

**EtaTSPipe**
Downstream system efficiency (t-s)

Type float
Read Only No

**EtaTT**
System efficiency (t-t)

Type float
Read Only No

**HeadRise**
User input, total head rise

Type Quantity
Read Only No

**HtrIn**
User input, hub/tip rotor inlet

Type float
Read Only No

**HtrOut**
User input, hub/tip rotor outlet

Type float
Read Only No
**HubLoadParam**
User input, hub loading parameter

**Type** float
**Read Only** No

**HubVelFactor**
User input, hub velocity deficit factor

**Type** float
**Read Only** No

**HubX**
Hub annulus X-coords

**Type** List<Quantity>
**Read Only** No

**HubY**
Hub annulus Y-coords

**Type** List<Quantity>
**Read Only** No

**IGV**
User input, IGV option

**Type** bool
**Read Only** No

**IGVhubThickX**
IGV hub thickness X-coord

**Type** List<float>
**Read Only** No

**IGVhubThickY**
IGV hub thickness Y-coord

**Type** List<Quantity>
**Read Only** No
**IGVleadingX**

IGV leading edge X-coords

**Type**  
List<Quantity>

**Read Only**  
No

**IGVleadingY**

IGV leading edge Y-coords

**Type**  
List<Quantity>

**Read Only**  
No

**IGVshrThickX**

IGV shroud thickness X-coord

**Type**  
List<float>

**Read Only**  
No

**IGVshrThickY**

IGV shroud thickness Y-coord

**Type**  
List<Quantity>

**Read Only**  
No

**IGVthetaLE**

IGV leading edge theta

**Type**  
List<float>

**Read Only**  
No

**IGVtrailingX**

IGV trailing edge X-coords

**Type**  
List<Quantity>

**Read Only**  
No

**IGVtrailingY**

IGV trailing edge Y-coords

**Type**  
List<Quantity>

**Read Only**  
No
**ImperialUnits**

User input, Imperial units option

*Type*  
bool

*Read Only*  
No

**InnerIter**

User input, number of inner loop design calculation iterations

*Type*  
int

*Read Only*  
No

**Layer1**

Intermediate spanwise layer1 for Export

*Type*  
bool

*Read Only*  
No

**Layer2**

Intermediate spanwise layer2 for Export

*Type*  
bool

*Read Only*  
No

**Layer3**

Intermediate spanwise layer3 for Export

*Type*  
bool

*Read Only*  
No

**LoadHub**

Rotor hub loading

*Type*  
float

*Read Only*  
No

**LoadMean**

Rotor mean loading

*Type*  
float

*Read Only*  
No
**MassFlow**

User input, mass flow rate

*Type*  
Quantity

*Read Only*  
No

**MaxLoadHub**

Rotor maximum hub loading

*Type*  
float

*Read Only*  
No

**MaxLoadMean**

Rotor maximum mean loading

*Type*  
float

*Read Only*  
No

**MixLoss**

User input, downstream mixing losses

*Type*  
float

*Read Only*  
No

**NMain**

Number of blades in each row

*Type*  
List<int>

*Read Only*  
No

**OGV**

User input, OGV option

*Type*  
bool

*Read Only*  
No

**OGVhubThickX**

OGV hub thickness X-coord

*Type*  
List<float>

*Read Only*  
No
**OGVhubThickY**

OGV hub thickness Y-coord

**Type** List<Quantity>

**Read Only** No

**OGVleadingX**

OGV leading edge X-coords

**Type** List<Quantity>

**Read Only** No

**OGVleadingY**

OGV leading edge Y-coords

**Type** List<Quantity>

**Read Only** No

**OGVshrThickX**

OGV shroud thickness X-coord

**Type** List<float>

**Read Only** No

**OGVshrThickY**

OGV shroud thickness Y-coord

**Type** List<Quantity>

**Read Only** No

**OGVthetaLE**

OGV leading edge theta

**Type** List<float>

**Read Only** No

**OGVtrailingX**

OGV trailing edge X-coords

**Type** List<Quantity>

**Read Only** No
**OGVtrailingY**

OGV trailing edge Y-coords

*Type*  
List<Quantity>

*Read Only*  
No

**Omega**

User input, rotational speed

*Type*  
Quantity

*Read Only*  
No

**OuterIter**

User input, number of outer loop design calculation iterations

*Type*  
int

*Read Only*  
No

**Pdyn**

Outlet dynamic pressure

*Type*  
Quantity

*Read Only*  
No

**PdynPipe**

Downstream dynamic pressure

*Type*  
Quantity

*Read Only*  
No

**PhiHub**

Rotor hub flow coefficient

*Type*  
float

*Read Only*  
No

**PhiMean**

Rotor mean flow coefficient

*Type*  
float

*Read Only*  
No
**Power**

Power

**Type**  
Quantity

**Read Only**  
No

**RatioIGV**

User input, IGV aspect ratio

**Type**  
float

**Read Only**  
No

**RatioOGV**

User input, OGV aspect ratio

**Type**  
float

**Read Only**  
No

**RatioRotor**

User input, Rotor aspect ratio

**Type**  
float

**Read Only**  
No

**RotorHubThickX**

Rotor hub thickness X-coord

**Type**  
List<float>

**Read Only**  
No

**RotorHubThickY**

Rotor hub thickness Y-coord

**Type**  
List<Quantity>

**Read Only**  
No

**RotorLeadingX**

Rotor leading edge X-coords

**Type**  
List<Quantity>

**Read Only**  
No
**RotorLeadingY**

Rotor leading edge Y-coords

**Type**  List<Quantity>

**Read Only**  No

**RotorShrThickX**

Rotor shroud thickness X-coord

**Type**  List<float>

**Read Only**  No

**RotorShrThickY**

Rotor shroud thickness Y-coord

**Type**  List<Quantity>

**Read Only**  No

**RotorThetaLE**

Rotor leading edge theta

**Type**  List<float>

**Read Only**  No

**RotorTrailingX**

Rotor trailing edge X-coords

**Type**  List<Quantity>

**Read Only**  No

**RotorTrailingY**

Rotor trailing edge Y-coords

**Type**  List<Quantity>

**Read Only**  No

**Sc90MaxIter**

User input, maximum number of solver iterations

**Type**  int

**Read Only**  No
**Sc90Relax**
User input, solver relaxation factor

**Type**  
float

**Read Only**  
No

**Sc90Tol**
User input, solver tolerance

**Type**  
float

**Read Only**  
No

**ShrX**
Shroud annulus X-coords

**Type**  
List<Quantity>

**Read Only**  
No

**ShrY**
Shroud annulus Y-coords

**Type**  
List<Quantity>

**Read Only**  
No

**Slunits**
User input, SI units option

**Type**  
bool

**Read Only**  
No

**Span**
Spanwise fractions

**Type**  
List<float>

**Read Only**  
No

**StagPressure**
User input, inlet stagnation pressure

**Type**  
Quantity

**Read Only**  
No
**StagTemp**

User input, inlet stagnation temperature

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
<th>Read Only</th>
<th>No</th>
</tr>
</thead>
</table>

**Torque**

Torque

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
<th>Read Only</th>
<th>No</th>
</tr>
</thead>
</table>

**TrimIGV**

User input, IGV profile trim

<table>
<thead>
<tr>
<th>Type</th>
<th>float</th>
<th>Read Only</th>
<th>No</th>
</tr>
</thead>
</table>

**TrimOGV**

User input, OGV profile trim

<table>
<thead>
<tr>
<th>Type</th>
<th>float</th>
<th>Read Only</th>
<th>No</th>
</tr>
</thead>
</table>

**TrimRotor**

User input, Rotor profile trim

<table>
<thead>
<tr>
<th>Type</th>
<th>float</th>
<th>Read Only</th>
<th>No</th>
</tr>
</thead>
</table>

**VanesIGV**

User input, IGV number of vanes

<table>
<thead>
<tr>
<th>Type</th>
<th>int</th>
<th>Read Only</th>
<th>No</th>
</tr>
</thead>
</table>

**VanesOGV**

User input, OGV number of vanes

<table>
<thead>
<tr>
<th>Type</th>
<th>int</th>
<th>Read Only</th>
<th>No</th>
</tr>
</thead>
</table>
**VanesRotor**

User input, Rotor number of vanes

**Type**  int

**Read Only**  No

**VistaAFDTitle**

Editor Title

**Type**  string

**Read Only**  No

---

**Vista AFD Meanline**

This container holds Meanline data for an instance of Vista AFD.

**Methods**

**Edit**

This command class launches the Vista popup GUI.

**GetVistaAFDMeanlineProperties**

This query takes a container reference and returns the Data Entity which contains user settings and properties for the VistaAFD Meanline container.

**Return**

Reference to the requested Data Entity

**Type**  DataReference

**Import**

This command imports Vista data into the Blade Design cell from an existing BladeGen file. If no appropriate Vista data is found in the specified BladeGen file, an error message is generated.

```python
    template1 = GetTemplate(TemplateName="VistaCPD")
    system1 = template1.CreateSystem()
    bladeDesign1 = system1.GetContainer(ComponentName="Blade Design")
    bladeDesign1.Import(FilePath="myfilepath/pump.bgd")
```

**Required Arguments**

**FileName**  Name, and path, of the BladeGen file to be imported.

**Type**  string
# Data Entities

**VistaAFDMeanline**

Meanline represents a VistaAFD meanline calculation as an initial 1D design

## Properties

### Alpha1

User input, IGV exit angle

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
<th>Read Only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>

### Alpha3

User input, OGV exit angle

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
<th>Read Only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>

### AlphaOGVHub

OGV hub gas exit angle

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
<th>Read Only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>

### AlphaOGVMean

OGV mean gas exit angle

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
<th>Read Only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>

### AlphaRotHub

Rotor hub gas exit angle

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
<th>Read Only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>

### AlphaRotMean

Rotor mean gas exit angle

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
<th>Read Only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>
DeHallerOGVHub

OGV hub DeHaller number

**Type**  
float

**Read Only**  
No

DeHallerOGVMean

OGV mean DeHaller number

**Type**  
float

**Read Only**  
No

DeHallerRotHub

Rotor hub DeHaller number

**Type**  
float

**Read Only**  
No

DeHallerRotMean

Rotor mean DeHaller number

**Type**  
float

**Read Only**  
No

Diameter

User input, outer diameter

**Type**  
Quantity

**Read Only**  
No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  
string

**Read Only**  
No

Eta

Aerodynamic efficiency

**Type**  
float

**Read Only**  
No
**EtaInput**

User input, efficiency estimate

- **Type**: float
- **Read Only**: No

**EtaTS**

System efficiency (t-s)

- **Type**: float
- **Read Only**: No

**EtaTSPipe**

Downstream system efficiency (t-s)

- **Type**: float
- **Read Only**: No

**EtaTT**

System efficiency (t-t)

- **Type**: float
- **Read Only**: No

**HeadRise**

User input, total head rise

- **Type**: Quantity
- **Read Only**: No

**HtrIn**

User input, hub/tip rotor inlet

- **Type**: float
- **Read Only**: No

**HtrOut**

User input, hub/tip rotor outlet

- **Type**: float
- **Read Only**: No
**HubLoadParam**
User input, hub loading parameter

**Type**  float
**Read Only**  No

**HubVelFactor**
User input, hub velocity deficit factor

**Type**  float
**Read Only**  No

**HubX**
Hub annulus X-coords

**Type**  List<Quantity>
**Read Only**  No

**HubY**
Hub annulus Y-coords

**Type**  List<Quantity>
**Read Only**  No

**IGV**
User input, IGV option

**Type**  bool
**Read Only**  No

**ImperialUnits**
User input, Imperial units option

**Type**  bool
**Read Only**  No

**LoadHub**
Rotor hub loading

**Type**  float
**Read Only**  No
**LoadMean**

Rotor mean loading

**Type** float

**Read Only** No

**MassFlow**

User input, mass flow rate

**Type** Quantity

**Read Only** No

**MaxLoadHub**

Rotor maximum hub loading

**Type** float

**Read Only** No

**MaxLoadMean**

Rotor maximum mean loading

**Type** float

**Read Only** No

**MixLoss**

User input, downstream mixing losses

**Type** float

**Read Only** No

**OGV**

User input, OGV option

**Type** bool

**Read Only** No

**Omega**

User input, rotational speed

**Type** Quantity

**Read Only** No
**Pdyn**
Outlet dynamic pressure

**Type**  Quantity
**Read Only**  No

**PdynPipe**
Downstream dynamic pressure

**Type**  Quantity
**Read Only**  No

**PhiHub**
Rotor hub flow coefficient

**Type**  float
**Read Only**  No

**PhiMean**
Rotor mean flow coefficient

**Type**  float
**Read Only**  No

**Power**
Power

**Type**  Quantity
**Read Only**  No

**RatioIGV**
User input, IGV aspect ratio

**Type**  float
**Read Only**  No

**RatioOGV**
User input, OGV aspect ratio

**Type**  float
**Read Only**  No
**RatioRotor**

User input, Rotor aspect ratio

*Type* float

*Read Only* No

**ShrX**

Shroud annulus X-coords

*Type* List<Quantity>

*Read Only* No

**ShrY**

Shroud annulus Y-coords

*Type* List<Quantity>

*Read Only* No

**Slunits**

User input, SI units option

*Type* bool

*Read Only* No

**StagPressure**

User input, inlet stagnation pressure

*Type* Quantity

*Read Only* No

**StagTemp**

User input, inlet stagnation temperature

*Type* Quantity

*Read Only* No

**Torque**

Torque

*Type* Quantity

*Read Only* No
**TrimIGV**
User input, IGV profile trim

**Type**       float

**Read Only**  No

**TrimOGV**
User input, OGV profile trim

**Type**       float

**Read Only**  No

**TrimRotor**
User input, Rotor profile trim

**Type**       float

**Read Only**  No

**VanesIGV**
User input, IGV number of vanes

**Type**       int

**Read Only**  No

**VanesOGV**
User input, OGV number of vanes

**Type**       int

**Read Only**  No

**VanesRotor**
User input, Rotor number of vanes

**Type**       int

**Read Only**  No

**VistaAFDTTitle**
Editor Title

**Type**       string

**Read Only**  No
Vista CCD
This container holds Analysis data for an instance of Vista CCD.

Methods

CreateBladeDesign
This command class creates a new BladeGen model. An up-to-date BladeGen cell appears on the project schematic containing the new model.

Optional Arguments

Beta  
Option to use the beta BladeGen template

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>False</td>
</tr>
</tbody>
</table>

CreateGeometry
This command class creates a new BladeEditor model. An up-to-date Geometry cell appears on the project schematic containing the new model.

CreateThroughflow
This command class creates a new throughflow system. The system, comprising Geometry, Setup, Solution and Results cells, is created on the project schematic and is updated performing the throughflow analysis automatically.

Optional Arguments

UseBladegen  
Indicates whether to use a Bladegen cell or a BladeEditor(Geometry) cell

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>False</td>
</tr>
</tbody>
</table>

CreateTurboflow
This command class creates a new Turbomachinery Fluid Flow system. The new system appears on the project schematic containing Geometry (BladeGen or BladeEditor), Turbo Mesh, Setup, Solution and Results cells. The Geometry, Turbo Mesh and Setup cells are automatically updated, leaving the CFD Solution 'ready to run'.

Optional Arguments

UseBladegen  
Indicates whether to use a Bladegen cell or a BladeEditor(Geometry) cell

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>False</td>
</tr>
</tbody>
</table>
Edit

This command class launches the Vista popup GUI.

GetVistaCCDBladeDesignProperties

This query takes a container reference and returns the Data Entity which contains user settings and properties for the VistaCCD Blade Design container.

Return  
Reference to the requested Data Entity  
Type  
DataReference

Import

This command imports Vista data into the Blade Design cell from an existing BladeGen file. If no appropriate Vista data is found in the specified BladeGen file, an error message is generated.

```python
template1 = GetTemplate(TemplateName="VistaCPD")
system1 = template1.CreateSystem()
bladeDesign1 = system1.GetContainer(ComponentName="Blade Design")
bladeDesign1.Import(FilePath="myfilepath/pump.bgd")
```

Required Arguments

FileName  
Name, and path, of the BladeGen file to be imported.  
Type  
string

Data Entities

VistaCCDBladeDesign

This data entity provides access to the properties which define the VistaCCD project. This includes both the input and the results properties.

Properties

Acentric

This property specifies the acentric factor for the working fluid. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

Type  
float  
Read Only  
No

Alpha3

This property specifies the flow angle at the impeller inlet in the absolute reference frame.

Type  
Quantity  
Read Only  
No
**Alpha5rms**

This property reports the absolute flow angle at the impeller trailing edge.

**Type** Quantity

**Read Only** No

**AnChkRatio**

This property reports the annulus choke ratio.

**Type** float

**Read Only** No

**B5**

This property reports the axial distance between hub and shroud at the impeller trailing edge (tip width).

**Type** Quantity

**Read Only** No

**Beta5rms**

This property reports the relative flow angle at the impeller trailing edge.

**Type** Quantity

**Read Only** No

**BetaBlade3HubUser**

This property specifies the hub leading edge blade angle. Note that this is NOT available when the StackingOption is set to 'Radial'.

**Type** Quantity

**Read Only** No

**BetaBlade3ShrUser**

This property specifies the impeller shroud leading edge blade angle. Note that this is only available when the ShroudDiameterOption is set to 'Angle'.

**Type** Quantity

**Read Only** No

**BetaBlade5**

This property specifies the impeller backsweep angle.

**Type** Quantity
**BetaBladeLEhub**

This property reports the blade angle at the impeller leading edge hub location.

- **Type**: Quantity
- **Read Only**: No

**BetaBladeLErms**

This property reports the blade angle at the impeller leading edge meanline location.

- **Type**: Quantity
- **Read Only**: No

**BetaBladeLEshr**

This property reports the blade angle at the impeller leading edge shroud location.

- **Type**: Quantity
- **Read Only**: No

**BetaLEhub**

This property reports the relative flow angle at the impeller leading edge hub location.

- **Type**: Quantity
- **Read Only**: No

**BetaLERms**

This property reports the relative flow angle at the impeller leading edge meanline location.

- **Type**: Quantity
- **Read Only**: No

**BetaLEshr**

This property reports the relative flow angle at the impeller leading edge shroud location.

- **Type**: Quantity
- **Read Only**: No

**BMunitsOption**

This property specifies the units used when creating a new BladeGen/BladeEditor model. Note that this is independent of the units used in the VistaCCD popup GUI.

Available options:
This property reports the impeller choke ratio.

**Type**: float

**Read Only**: No

**ChokeUser**

This property specifies the impeller choke ratio. Note that this is only available when the IncidenceOption is set to 'choke'.

**Type**: float

**Read Only**: No

**ClearanceOption**

This property specifies impeller tip clearance is specified. 'Ratio' indicates that the tip clearance is specified as a fraction of the tip width 'User' specifies that the clearance will be defined directly by the user.

Available options:

- Ratio
- User

**Type**: ClearanceType

**Read Only**: No

**ClearRatio**

This property reports the axial tip clearance ratio of the impeller.

**Type**: float

**Read Only**: No

**ClearRatioUser**

This property specifies the ratio of the impeller tip clearance to the tip width. Note that this is only available when ClearanceOption is set to 'Ratio'.

**Type**: float
**Read Only** No

**ClearUser**

This property specifies the value of the impeller tip clearance. Note that this is only available when ClearanceOption is set to 'User'.

**Type** Quantity

**Read Only** No

**CorrelationOption**

This property specifies the correlation used to calculate the stage efficiency. Note that this is only available when the EfficiencyOption is set to 'Correlation'.

Available options:

CaseyRobinson
CaseyMarty
Rodgers

**Type** EtaCorrelType

**Read Only** No

**Cp_A0**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the constant component (coefficient of $T^0$) of the lower temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

**Type** double

**Read Only** No

**Cp_A1**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the coefficient of $T^1$ of the lower temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

**Type** double

**Read Only** No

**Cp_A2**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the coefficient of $T^2$ of the lower temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.
For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the coefficient of $T^3$ of the lower temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

**Cp_A3**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the coefficient of $T^4$ of the lower temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

**Cp_A4**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the coefficient of $T^5$ of the lower temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

**Cp_A5**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the coefficient of $T^6$ of the lower temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

**Cp_A6**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the coefficient of $T^7$ of the lower temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.
**Type** double

**Read Only** No

**Cp_Amax**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the maximum temperature limit for which the lower temperature range polynomial is applicable. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

**Type** float

**Read Only** No

**Cp_Amin**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the minimum temperature limit for which the lower temperature range polynomial is applicable. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

**Type** float

**Read Only** No

**Cp_B0**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the constant component (coefficient of T^0) of the upper temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

**Type** double

**Read Only** No

**Cp_B1**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the coefficient of T^1 of the upper temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

**Type** double

**Read Only** No

**Cp_B2**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the coefficient of T^2 of the upper temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.
**Type** double

**Read Only** No

**Cp_B3**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the coefficient of $T^3$ of the upper temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

**Type** double

**Read Only** No

**Cp_B4**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the coefficient of $T^4$ of the upper temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

**Type** double

**Read Only** No

**Cp_B5**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the coefficient of $T^5$ of the upper temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

**Type** double

**Read Only** No

**Cp_B6**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the coefficient of $T^6$ of the upper temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

**Type** double

**Read Only** No

**Cp_B7**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the coefficient of $T^7$ of the upper temperature range polynomial. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.
Type: double

**Read Only** No

**Cp_Bmax**

For a user-defined real gas, the specific heat capacity is specified as a polynomial function of temperature over two temperature ranges. This property specifies the maximum temperature limit for which the upper temperature range polynomial is applicable. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

Type: float

**Read Only** No

**CriticalPressure**

This property specifies the critical pressure for the working fluid. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

Type: Quantity

**Read Only** No

**CriticalTemp**

This property specifies the critical temperature for the working fluid. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

Type: Quantity

**Read Only** No

**CriticalVol**

This property specifies the critical volume for the working fluid. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Real'.

Type: Quantity

**Read Only** No

**D3Hub**

This property specifies the hub inlet diameter for the impeller.

Type: Quantity

**Read Only** No

**D3ShrUser**

This property specifies the impeller shroud diameter at the leading edge. Note that this is only available when the ShroudDiameterOption is set to 'Diameter'.

Type: Quantity
**Read Only**  No

**D5**

This property reports the diameter at the impeller trailing edge (tip diameter).

*Type*  Quantity

*Read Only*  No

**Diffuser**

This property specifies whether the diffuser section is vaned or vaneless.

Available options:

- Vaned
- Vaneless

*Type*  DiffuserType

*Read Only*  No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type*  string

*Read Only*  No

**DLEhub**

This property reports the diameter at the impeller leading edge hub location.

*Type*  Quantity

*Read Only*  No

**DLErms**

This property reports the diameter at the impeller leading edge meanline location.

*Type*  Quantity

*Read Only*  No

**DLEshr**

This property reports the diameter at the impeller leading edge shroud location.

*Type*  Quantity

*Read Only*  No
**EfficiencyOption**

This property specifies whether to use a correlation to automatically calculate the compressor stage efficiency, or to use a user-defined efficiency.

Available options:

- Correlation
- User

```python
template1 = GetTemplate(TemplateName="VistaCCD")
system1 = template1.CreateSystem()
bladeDesign1 = system1.GetContainer(ComponentName="Blade Design")
vistaCCDProperties1 = bladeDesign1.GetVistaCCDBladeDesignProperties()
vistaCCDProperties1.EfficiencyOption = "Correlation"
```

<table>
<thead>
<tr>
<th>Type</th>
<th>EtaType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**Etalsen**

This property reports the isentropic efficiency for the compressor stage.

- Type | float |
- Read Only | No |

**EtalsenImp**

This property reports the isentropic efficiency for the compressor impeller.

- Type | float |
- Read Only | No |

**EtalsenImpUser**

This property specifies the user defined impeller isentropic efficiency. Note that this is only available when the ImpellerEfficiencyOption is set to 'User'.

- Type | float |
- Read Only | No |

**EtalsenUser**

This property specifies the user defined stage isentropic efficiency. Note that this is only available when the UserEfficiencyOption is set to 'Isentropic'.

- Type | float |
- Read Only | No |

**EtaPoly**

This property reports the polytropic efficiency for the compressor stage.
**Type**  
**float**

**Read Only**  
**No**

**EtaPolyImp**

This property reports the polytropic efficiency for the compressor impeller.

**Type**  
**float**

**Read Only**  
**No**

**EtaPolyUser**

This property specifies the user defined stage polytropic efficiency. Note that this is only available when the UserEfficiencyOption is set to 'Polytropic'.

**Type**  
**float**

**Read Only**  
**No**

**Gamma**

This property reports the ratio of specific heats of the working fluid.

**Type**  
**float**

**Read Only**  
**No**

**GammaUser**

This property specifies the ratio of specific heats for the working fluid. Note that this is only available when MaterialPropsOption is set to 'User' and the GasModelOption is set to 'Ideal'.

**Type**  
**float**

**Read Only**  
**No**

**GasModelOption**

This property specifies whether to treat the working fluid as an Ideal or a Real gas.

Available options:

- Ideal
- Real

**Type**  
**GasModelType**

**Read Only**  
**No**

**GeometryStyle**

This property specifies the approach taken when creating a new Geometry model from a successful VistaCCD calculation.
Available options:

- Interactive
- Parametric

**Type**  
**GeometryStyleType**

**Read Only**  
No

**H05**

This property reports the total enthalpy at the impeller trailing edge.

**Type**  
**Quantity**

**Read Only**  
No

**H0LE**

This property reports the stagnation enthalpy at the impeller leading edge.

**Type**  
**Quantity**

**Read Only**  
No

**Impeller**

This property specifies whether the impeller is unshrouded or shrouded.

Available options:

- Unshrouded
- Shrouded

**Type**  
**ImpellerType**

**Read Only**  
No

**ImpellerEfficiencyOption**

This property specifies whether to automatically calculate the impeller efficiency by linking this to the stage efficiency, or to use a user-defined efficiency.

Available options:

- LinkToStage
- User

**Type**  
**EtaImpType**

**Read Only**  
No

**ImpellerLength**

This property specifies the impeller axial length to tip diameter ratio.
Type: float
Read Only: No

**ImpellerLengthOption**

This property specifies whether the impeller length ratio is calculated automatically, or specified by the user.

Available options:
- Automatic
- User

Type: ImpellerLengthType
Read Only: No

**ImpellerLengthUserOpt**

This property specifies whether the impeller axial length ratio will be defined by the user.

Type: bool
Read Only: No

**IncidenceOption**

This property specifies the method used to calculate the incidence at the impeller shroud. The incidence may be either specified directly or calculated using the specified choke ratio.

Available options:
- incidence
- choke

Type: IncidenceType
Read Only: No

**IncLEhub**

This property reports the impeller incidence at the hub location.

Type: Quantity
Read Only: No

**IncLERms**

This property reports the impeller incidence at the meanline location.

Type: Quantity
Read Only: No
**IncLEshr**

This property reports the impeller incidence at the shroud location.

- **Type**: Quantity
- **Read Only**: No

**IncShrUser**

This property specifies the incidence at the impeller shroud. Note that this is only available when the IncidenceOption is set to 'incidence'.

- **Type**: Quantity
- **Read Only**: No

**LEInclination**

This property reports the leading edge angle of inclination in the meridional plane.

- **Type**: Quantity
- **Read Only**: No

**LEInclinationUser**

This property specifies the inclination of the leading edge relative to a radial line in the meridional view.

- **Type**: Quantity
- **Read Only**: No

**Loading**

This property reports the impeller loading parameter (delH/U^2).

- **Type**: float
- **Read Only**: No

**M5rms**

This property reports the absolute Mach number at the impeller trailing edge.

- **Type**: float
- **Read Only**: No

**MachU5**

This property reports the blade Mach number at the impeller trailing edge (tip Mach number).

- **Type**: float
- **Read Only**: No
**MassFlow**

This property specifies the design point mass flow rate passing through the compressor stage.

**Type** Quantity

**Read Only** No

**MaterialNameSelection**

This property specifies the name of the working fluid, as selected from the database. Note that this is only available when the MaterialPropsOption is set to 'Database'.

Available options:

- air
- carbon_dioxide
- hydrogen
- methane
- nitrogen
- oxygen
- parahydrogen
- propylene
- R123
- R125
- R134a
- R141b
- R142b
- R245fa
- water

```python
template1 = GetTemplate(TemplateName="VistaCCD")
system1 = template1.CreateSystem()
bladeDesign1 = system1.GetContainer(ComponentName="Blade Design")
vistaCCDProperties1 = bladeDesign1.GetVistaCCDBladeDesignProperties()
vistaCCDProperties1.MaterialPropsOption = "Database"
vistaCCDProperties1.MaterialNameSelection = "nitrogen"
```

**Type** MaterialNamesList

**Read Only** No

**MaterialPropsOption**

This property specifies whether the working fluid properties are chosen from the materials database or specified directly by the user.

Available options:

- Database
- User

**Type** MaterialPropsType

**Read Only** No
**MerVelGrad**

This property specifies the gradient of the velocity profile from hub to shroud at the impeller leading edge. The gradient is set using the ratio of the meridional velocity at the shroud leading edge radius to that at the average leading edge radius.

**Type** float

**Read Only** No

**MrelLEhub**

This property reports the relative Mach number at the impeller leading edge hub location.

**Type** float

**Read Only** No

**MrelLErms**

This property reports the relative Mach number at the impeller leading edge meanline location.

**Type** float

**Read Only** No

**MrelLEshr**

This property reports the relative Mach number at the impeller leading edge shroud location.

**Type** float

**Read Only** No

**MrmsLE**

This property reports the absolute Mach number at the impeller leading edge meanline location.

**Type** float

**Read Only** No

**Mu**

This property reports the dynamic viscosity of the working fluid.

**Type** Quantity

**Read Only** No

**MuUser**

This property specifies the dynamic viscosity of the working fluid.

**Type** Quantity
**Read Only**  No

**NMMain**

This property specifies the number of impeller main vanes.

**Type**  \text{int}

**Read Only**  No

**NormalToHubLE**

This property specifies that the main impeller blade leading is normal to the hub curve.

**Type**  \text{bool}

**Read Only**  No

**Ns**

This property reports the specific speed of the impeller.

**Type**  \text{float}

**Read Only**  No

**NSplit**

This property specifies the number of impeller splitter vanes. Note that this MUST be a multiple of the number of impeller main vanes.

**Type**  \text{int}

**Read Only**  No

**Nu**

This property reports the kinematic viscosity of the working fluid.

**Type**  \text{Quantity}

**Read Only**  No

**NuUser**

This property specifies the kinematic viscosity of the working fluid.

**Type**  \text{Quantity}

**Read Only**  No

**Omega**

This property specifies the design point rotational speed of the impeller.

**Type**  \text{Quantity}
**Read Only**  No

**P05rms**

This property reports the total pressure at the impeller trailing edge.

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**P5rms**

This property reports the static pressure at the impeller trailing edge.

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**PIF**

This property reports the power input factor of the compressor.

<table>
<thead>
<tr>
<th>Type</th>
<th>float</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**PIFOption**

This property specifies whether the power input factor is calculated using a correlation, or specified by the user.

Available options:

- Correlation
- User

<table>
<thead>
<tr>
<th>Type</th>
<th>PIFType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**PIFUser**

This property specifies the user defined power input factor. Note that this is only available when the PIFOption is set to 'User'.

<table>
<thead>
<tr>
<th>Type</th>
<th>float</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**Power**

This property reports the impeller power.

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>
**PressureRatio**

This property specifies the design point total-to-total pressure ratio for the compressor stage.

Type: float

Read Only: No

**PreswirlOption**

This property specifies how the radial distribution of the preswirl angle is calculated.

Available options:
- constant
- free
- forced
- linear

Type: PreswirlType

Read Only: No

**RakeAngle**

This property specifies the impeller rake angle (trailing edge lean angle).

Type: Quantity

Read Only: No

**Reb5**

This property reports the Reynolds number based on the impeller tip width dimension.

Type: float

Read Only: No

**ReCorrectOpt**

This property specifies if the Reynolds number correction is to be made to the stage efficiency correlation. Note that this is only available when the EfficiencyOption is set to 'Correlation'.

Type: bool

Read Only: No

**Red5**

This property reports the Reynolds number based on the impeller tip diameter dimension.

Type: float

Read Only: No
**RelVelRatio**

This property specifies the ratio of the relative velocity at the trailing edge to that at the leading edge shroud location.

- **Type**: float
- **Read Only**: No

**RelVelRatioMod**

This property reports the ratio of the rms relative velocity at the trailing edge to the shroud relative velocity at the leading edge.

- **Type**: float
- **Read Only**: No

**RGas**

This property reports the specific gas constant of the working fluid.

- **Type**: Quantity
- **Read Only**: No

**RUser**

This property specifies the specific gas constant for the working fluid. Note that this is only available when MaterialPropsOption is set to 'User'.

- **Type**: Quantity
- **Read Only**: No

**S5**

This property reports the specific entropy at the impeller trailing edge.

- **Type**: Quantity
- **Read Only**: No

**ShrLELoc**

This property specifies the main impeller blade leading edge location on the shroud.

- **Type**: float
- **Read Only**: No

**ShroudDiameterOption**

This property specifies the method used to calculate the impeller shroud diameter at the leading edge. ‘Diameter’ allows the diameter to be directly specified. ‘Angle’ indicates that the diameter will be calcu-
lated from the shroud leading edge blade angle. 'Optimum' calculates the diameter such that the relative Mach number at the shroud leading edge is minimised.

Available options:

- Diameter
- Angle
- Optimum

**Type** ShroudDiameterType

**Read Only** No

**SLE**

This property reports the specific entropy at the impeller leading edge.

**Type** Quantity

**Read Only** No

**StackingOption**

This property specifies the method used to calculate the leading edge blade angles. Using the radial approach both hub and meanline leading edge blade angles are calculated from the shroud leading edge blade angle. Using either tangential or sine based approaches, the hub leading edge blade angle is user defined and the meanline leading edge blade angle is interpolated from the hub and shroud blade angles.

Available options:

- Radial
- Tan
- Sin

**Type** StackingType

**Read Only** No

**StagPressure**

This property specifies the design point stagnation pressure at the inlet to the compressor stage.

**Type** Quantity

**Read Only** No

**StagTemp**

This property specifies the design point stagnation temperature at the inlet to the compressor stage.

**Type** Quantity

**Read Only** No
**SurfaceFinish**

This property specifies the surface finish of the impeller. The surface roughness has a secondary effect on the calculated efficiency.

Available options:

- Machined
- Cast

**Type** \( \text{RoughnessType} \)

**Read Only** \( \text{No} \)

**SzrFlowCoeff**

This property reports the impeller flow coefficient.

**Type** \( \text{float} \)

**Read Only** \( \text{No} \)

**T05rms**

This property reports the total temperature at the impeller trailing edge.

**Type** \( \text{Quantity} \)

**Read Only** \( \text{No} \)

**ThkHub**

This property specifies the hub vane normal thickness.

**Type** \( \text{Quantity} \)

**Read Only** \( \text{No} \)

**ThkShr**

This property specifies the shroud vane normal thickness.

**Type** \( \text{Quantity} \)

**Read Only** \( \text{No} \)

**ThroatAreaLE**

This property reports the throat area at the impeller leading edge.

**Type** \( \text{Quantity} \)

**Read Only** \( \text{No} \)
**TipCorrectOpt**

This property specifies if the tip clearance and shroud correction is to be made to the stage efficiency correlation. Note that this is only available when the EfficiencyOption is set to 'Correlation'.

**Type**  
bool

**Read Only**  
No

**U5**

This property reports the blade speed at the impeller trailing edge (tip speed).

**Type**  
Quantity

**Read Only**  
No

**UserEfficiencyOption**

This property specifies whether the user defined stage efficiency is isentropic or polytropic. Note that this is only available when the EfficiencyOption is set to 'User'.

Available options:

- Isentropic
- Polytropic

**Type**  
EtaUserType

**Read Only**  
No

**V5rms**

This property reports the absolute velocity at the impeller trailing edge.

**Type**  
Quantity

**Read Only**  
No

**ViscosityOption**

This property specifies the method used to set the viscosity of the working fluid. The viscosity may be calculated using Sutherland’s law, specified as a constant dynamic viscosity or as a constant kinematic viscosity. Note that this is only available when MaterialPropsOption is set to 'User'.

Available options:

- Sutherland
- Dynamic
- Kinematic

**Type**  
ViscosityType

**Read Only**  
No
**ViscosityOptionR145**

This property specifies the method used to set the kinematic viscosity of the working fluid. The kinematic viscosity may either be calculated using Sutherland's law for Air, or defined as a constant value by the user. Note that this is only available when MaterialPropsOption is set to 'User'.

Available options:

- Sutherland
- User

**VmLEhub**

This property reports the meridional velocity at the impeller leading edge hub location.

**Type** Quantity

**Read Only** No

**VmLErms**

This property reports the meridional velocity at the impeller leading edge meanline location.

**Type** Quantity

**Read Only** No

**VmLEshr**

This property reports the meridional velocity at the impeller leading edge shroud location.

**Type** Quantity

**Read Only** No

**VmRatioLE**

This property reports the ratio of the shroud meridional velocity to the RMS meridional velocity at the leading edge.

**Type** float

**Read Only** No

**VrmsLE**

This property reports the absolute velocity at the impeller leading edge meanline location.

**Type** Quantity

**Read Only** No
**VwLEhub**

This property reports the tangential velocity at the impeller leading edge hub location.

**Type** Quantity

**Read Only** No

**VwLERms**

This property reports the tangential velocity at the impeller leading edge meanline location.

**Type** Quantity

**Read Only** No

**VwLEshr**

This property reports the tangential velocity at the impeller leading edge shroud location.

**Type** Quantity

**Read Only** No

**VwRatioLE**

This property reports the ratio of the shroud swirl velocity to the RMS swirl velocity at the leading edge.

**Type** float

**Read Only** No

**VwRatioUser**

This property specifies the ratio of the inlet tangential velocity at the shroud to that at the meanline. Note that this property is only valid when the PreswirlOption is set to linear.

**Type** float

**Read Only** No

**W5rms**

This property reports the relative velocity at the impeller trailing edge.

**Type** Quantity

**Read Only** No
Vista CCM

This container holds Analysis data for an instance of Vista CCM.

Methods

Edit

This command class launches the Vista popup GUI.

GetVistaCCMBladeDesignProperties

This query takes a container reference and returns the Data Entity which contains user settings and properties for the VistaCCD Performance Map container.

Return

Reference to the requested Data Entity

Type

DataReference

Data Entities

VistaCCMBladeDesign

Setup represents a VistaCCM project definition.

Properties

Alpha3

Inlet angle

Type

Quantity

Read Only

No

B5

User input: impeller tip width

Type

Quantity

Read Only

No

BetaBlade5

User input: impeller blade exit angle

Type

Quantity

Read Only

No
**BetaBladeLEhub**

User input: hub vane angle at the leading edge

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**BetaBladeLEshr**

User input: shroud vane angle at the leading edge

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**ConditionsFromUpstream**

Flag to choose whether to update operating conditions from upstream cell or not

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**D5**

User input: impeller exit diameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

<table>
<thead>
<tr>
<th>Type</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**DLEhub**

User input: Hub diameter at the leading edge

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**DLEshr**

User input: shroud diameter at the leading edge

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>
**EffPolyData**

Results data: Range of polytropic efficiencies

- **Type**: `List<List<float>>`
- **Read Only**: No

**EtaPoly**

Polytropic efficiency

- **Type**: `float`
- **Read Only**: No

**GammaUser**

Ratio of specific heats

- **Type**: `float`
- **Read Only**: No

**GasModel**

Gas model

- **Type**: `int`
- **Read Only**: No

**ImperialUnits**

User input, Imperial units option

- **Type**: `bool`
- **Read Only**: No

**KDiff**

User input: Diffuser type (-1.0 = vaneless, 1.0 = vaned)

- **Type**: `float`
- **Read Only**: No

**KType**

User input: Machine type (-1.0 = process, 1.0 = turbocharger)

- **Type**: `float`
- **Read Only**: No
**ListSpeed**

Results data: Range of speeds

**Type**  List<float>

**Read Only**  No

**Loading**

Work factor

**Type**  float

**Read Only**  No

**MachU5**

Tip Mach number

**Type**  float

**Read Only**  No

**MassFlow**

Design point mass flow rate

**Type**  Quantity

**Read Only**  No

**MassFlowData**

Results data: Range of mass flow rates

**Type**  List<List<float>>

**Read Only**  No

**NMain**

User input: number of main vanes

**Type**  int

**Read Only**  No

**NSpeeds**

User input, number of speeds

**Type**  int

**Read Only**  No
**NSplit**

User input: number of splitter vanes

**Type** int

**Read Only** No

**PIF**

Power input factor

**Type** float

**Read Only** No

**PresRatioData**

Results data: Range of pressure ratios

**Type** List<List<float>>

**Read Only** No

**PressureRatio**

Design point pressure ratio

**Type** float

**Read Only** No

**RUser**

Gas constant

**Type** Quantity

**Read Only** No

**Slunits**

User input, SI units option

**Type** bool

**Read Only** No

**SpeedMax**

User input, maximum speed

**Type** Quantity

**Read Only** No
**SpeedMaxFixed**
User input, maximum speed

**Type** Quantity
**Read Only** No

**SpeedMin**
User input, minimum speed

**Type** Quantity
**Read Only** No

**SpeedMinFixed**
User input, minimum speed

**Type** Quantity
**Read Only** No

**StagPressure**
Inlet stagnation pressure

**Type** Quantity
**Read Only** No

**StagPressureFixed**
Inlet stagnation pressure

**Type** Quantity
**Read Only** No

**StagTemp**
Inlet stagnation temperature

**Type** Quantity
**Read Only** No

**StagTempFixed**
Inlet stagnation temperature

**Type** Quantity
**Read Only** No
**SzrFlowCoeff**

Flow coefficient

**ThkHub**

User input: Hub vane normal thickness

**ThkShr**

User input: shroud vane normal thickness

**ThroatArea**

User input: throat area

**TipMachInData**

Results data: Range of inlet tip Mach numbers

**TipMachOutData**

Results data: Range of outlet tip Mach numbers

**VistaCCMTitle**

Editor Title
Vista CPD
This container holds Analysis data for an instance of Vista CPD.

Methods

CreateBladeDesign
This command class creates a new BladeGen model. An up-to-date BladeGen cell appears on the project schematic containing the new model.

Optional Arguments

Beta  
Option to use the beta BladeGen template

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>False</td>
</tr>
</tbody>
</table>

CreateGeometry
This command class creates a new BladeEditor model. An up-to-date Geometry cell appears on the project schematic containing the new model.

CreateThroughflow
This command class creates a new throughflow system. The system, comprising Geometry, Setup, Solution and Results cells, is created on the project schematic and is updated performing the throughflow analysis automatically.

Optional Arguments

UseBladegen  
Indicates whether to use a Bladegen cell or a BladeEditor(Geometry) cell

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>False</td>
</tr>
</tbody>
</table>

CreateVoluteMesh
This command class creates a new pump volute geometry and mesh. An up-to-date Mesh system appears on the project schematic containing the new geometry and mesh cells.

Edit
This command class launches the Vista popup GUI.

GetVistaCPDbladeDesignProperties
This query takes a container reference and returns the Data Entity which contains user settings and properties for the VistaCPD Blade Design container.

Return  
Reference to the requested Data Entity
**Import**

This command imports Vista data into the Blade Design cell from an existing BladeGen file. If no appropriate Vista data is found in the specified BladeGen file, an error message is generated.

```python
template1 = GetTemplate(TemplateName="VistaCPD")
system1 = template1.CreateSystem()
bladeDesign1 = system1.GetContainer(ComponentName="Blade Design")
bladeDesign1.Import(FilePath="myfilepath/pump.bgd")
```

**Required Arguments**

- **FileName** Name, and path, of the BladeGen file to be imported.

  *Type* string

**Data Entities**

**VistaCPDBladeDesign**

This data entity provides access to the properties which define the VistaCPD project. This includes both the input and the results properties.

**Properties**

- **Alpha2**
  
  This property reports the absolute flow angle at the impeller trailing edge.

  *Type* Quantity

  *Read Only* Yes

- **AlphaIn**
  
  This property specifies the absolute flow angle, measured with respect to the tangential direction, at the leading edge of the pump impeller.

  *Type* Quantity

  *Read Only* No

- **AreaDiff**
  
  This property reports the volute diffuser exit area.

  *Type* Quantity

  *Read Only* Yes
**AspectRatio**

This property specifies the aspect ratio (height/width) of the rectangular volute cross section. This is valid when `VoluteStyleOpt` is 'Rectangular'.

**Type**  
(float)

**Read Only**  
No

**B2**

This property reports the hub to shroud distance at the impeller trailing edge (tip width).

**Type**  
(Quantity)

**Read Only**  
Yes

**B3**

This property reports the width of the volute inlet.

**Type**  
(Quantity)

**Read Only**  
Yes

**Beta1**

This property reports the relative flow angle at the impeller leading edge meanline section.

**Type**  
(Quantity)

**Read Only**  
Yes

**Beta1Blade**

This property reports the impeller leading edge blade angle at the meanline section.

**Type**  
(Quantity)

**Read Only**  
Yes

**Beta1BladeHub**

This property reports the impeller leading edge blade angle at the hub section.

**Type**  
(Quantity)

**Read Only**  
Yes

**Beta1BladeHubUser**

This property specifies the impeller leading edge blade angle at the hub, measured with respect to the tangential direction. This is valid when the `HubBeta1Opt` is set to 'User'.

**Type**  
(Quantity)
Read Only No

**Beta1BladeShr**
This property reports the impeller leading edge blade angle at the shroud section.

Type Quantity
Read Only Yes

**Beta1BladeShrUser**
This property specifies the impeller leading edge blade angle at the shroud, measured with respect to the tangential direction. This is valid when the ShrBeta1Opt is set to 'User'.

Type Quantity
Read Only No

**Beta1BladeUser**
This property specifies the impeller leading edge blade angle at the meanline, measured with respect to the tangential direction. This is valid when the HubBeta1Opt is set to 'User'.

Type Quantity
Read Only No

**Beta1Hub**
This property reports the relative flow angle at the impeller leading edge hub section.

Type Quantity
Read Only Yes

**Beta1Shr**
This property reports the relative flow angle at the impeller leading edge shroud section.

Type Quantity
Read Only Yes

**Beta2**
This property reports the relative flow angle at the impeller trailing edge.

Type Quantity
Read Only Yes

**Beta2Blade**
This property specifies the blade angle at the impeller trailing edge, measured with respect to the tangential direction.
**BMExportOption**

This property specifies whether the impeller is to be exported as an isolated impeller or coupled to a volute. The isolated impeller option provides for an extended exit diffuser to assist the analysis process. If the impeller is coupled to the volute, the exit diffuser is short to match with the volute inlet.

Available options:
- Isolated
- Coupled

**BMunitsOption**

This property specifies the units used when creating a new BladeGen/BladeEditor model. Note that this is independent of the units used in the VistaCPD popup GUI.

Available options:
- mm
- inches

**C2**

This property reports the absolute flow velocity at the impeller trailing edge.

**Cm1**

This property reports the meridional flow velocity at the impeller leading edge meanline section.

**Cm1Hub**

This property reports the meridional flow velocity at the impeller leading edge hub section.
**Cm1Shr**

This property reports the meridional flow velocity at the impeller leading edge shroud section.

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Cu1**

This property reports the tangential flow velocity at the impeller leading edge meanline section.

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Cu1Hub**

This property reports the tangential flow velocity at the impeller leading edge hub section.

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Cu1Shr**

This property reports the tangential flow velocity at the impeller leading edge shroud section.

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Cu2**

This property reports the tangential flow velocity at the impeller trailing edge.

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Cus**

This property reports the slip velocity at the impeller trailing edge. This is defined as the difference between the theoretical 'no-slip' tangential flow velocity and the true tangential flow velocity.

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**D1**

This property reports the diameter of the impeller leading edge at the meanline section.

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>Yes</td>
</tr>
</tbody>
</table>
**D1Hub**

This property reports the diameter of the impeller leading edge at the hub section.

**Type**  Quantity

**Read Only**  Yes

**D1Shr**

This property reports the diameter of the impeller leading edge at the shroud section.

**Type**  Quantity

**Read Only**  Yes

**D2**

This property reports the diameter at the impeller trailing edge meanline section (tip diameter).

**Type**  Quantity

**Read Only**  Yes

**D2Opt**

This property specifies the method used to set the impeller tip diameter. It may be calculated automatically, from a specified head coefficient or the value may be user defined.

Available options:

- Automatic
- HeadCoeff
- User

**Type**  TipDiamType

**Read Only**  No

**D2User**

This property specifies the impeller tip diameter. This is valid when the D2Opt is set to 'User'.

**Type**  Quantity

**Read Only**  No

**DEye**

This property reports the impeller shroud diameter at the eye of the impeller.

**Type**  Quantity

**Read Only**  Yes
**DEyeHub**

This property reports the impeller hub diameter at the eye of the impeller.

*Type*  Quantity

*Read Only*  Yes

**DiamDiff**

This property reports the hydraulic diameter of the volute diffuser exit.

*Type*  Quantity

*Read Only*  Yes

**DiamDiffUser**

This property specifies the value of the volute diffuser exit diameter. This is valid when UserDiamDiff is set to true.

*Type*  Quantity

*Read Only*  No

**DiffRatio**

This property reports the diffusion ratio of the impeller.

*Type*  float

*Read Only*  Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

*Type*  string

*Read Only*  No

**DShaft**

This property reports the impeller shaft diameter.

*Type*  Quantity

*Read Only*  Yes

**EfficiencyOption**

This property specifies how the impeller efficiencies are calculated.

With this set to 'Automatic' the efficiencies are calculated using empirical correlations. All other options require the specification of three of the efficiencies. The remaining value is calculated from those specified.
For example, specifying 'Hydraulic' indicates that the hydraulic efficiency will be calculated and the volumetric, mechanical and overall pump efficiencies must be specified.

Available options:

- Automatic
- Hydraulic
- Volumetric
- Mechanical
- Pump

**FlowCoeff**

This property reports the flow coefficient of the impeller.

- **Type**: float
- **Read Only**: Yes

**Head**

This property specifies the design point head rise for the impeller.

- **Type**: Quantity
- **Read Only**: No

**HeadCoeff**

This property reports the head coefficient of the impeller.

- **Type**: float
- **Read Only**: Yes

**HeadCoeffUser**

This property specifies the head coefficient, used to calculate the impeller tip diameter. This is valid when the D2Opt is set to 'HeadCoeff'.

- **Type**: float
- **Read Only**: No

**HeightDiff**

This property reports the volute diffuser exit height. In the case of a circular outlet, (elliptic volute cross section), this is the same as the hydraulic diameter.

- **Type**: Quantity
- **Read Only**: Yes
**HubBeta1Opt**

This property specifies the method used to set the impeller leading edge blade angles at the hub and meanline sections. These blade angles may be calculated relative to the leading edge blade angle at the shroud using either cosine or cotangent relationships, or they may be defined directly by the user.

Available options:

- Cos
- Cot
- User

**Type** HubLEBetaType

**Read Only** No

**HubInletDraft**

This property specifies the impeller hub inlet draft angle. This is defined as the angle between the hub and the horizontal line at the hub inlet.

**Type** Quantity

**Read Only** No

**HydEff**

This property reports the hydraulic efficiency of the impeller.

**Type** float

**Read Only** Yes

**HydEffUser**

This property specifies the impeller hydraulic efficiency. This is valid when the EfficiencyOption is set to 'Volumetric', 'Mechanical' or 'Pump'.

**Type** float

**Read Only** No

**Inc**

This property reports the incidence at the impeller leading edge meanline section.

**Type** Quantity

**Read Only** Yes

**IncHub**

This property reports the incidence at the impeller leading edge hub section.

**Type** Quantity
**Read Only**  Yes

**IncShr**
This property reports the incidence at the impeller leading edge shroud section.

**Type**  Quantity

**Read Only**  Yes

**IncShrUser**
This property specifies the angle of incidence for the impeller at the shroud. This is valid when the ShrBeta1Opt is set to 'Incidence'.

**Type**  Quantity

**Read Only**  No

**Ks**
This property reports the stability factor of the impeller.

**Type**  float

**Read Only**  Yes

**LengthDiff**
This property reports the volute diffuser axial length.

**Type**  Quantity

**Read Only**  Yes

**LengthDiffUser**
This property specifies the value of the volute diffuser axial length. This is valid when UserLengthDiff is set to true.

**Type**  Quantity

**Read Only**  No

**MechEff**
This property reports the mechanical efficiency of the impeller.

**Type**  float

**Read Only**  Yes

**MechEffUser**
This property specifies the impeller mechanical efficiency. This is valid when the EfficiencyOption is set to 'Hydraulic', 'Volumetric' or 'Pump'.
**Mer Vel Ratio**

This property specifies the gradient of the velocity profile from hub to shroud at the impeller leading edge. The gradient is set using the ratio of the meridional velocity at the shroud leading edge radius to that at the average leading edge radius.

**Min Diam Factor**

This property specifies the shaft minimum diameter factor. This is a 'factor of safety' applied to the shaft minimum diameter as calculated from the maximum allowable shear stress of the shaft.

**NPSH r**

This property reports the net positive suction head required (NPSHr) of the impeller.

**Nq**

This property reports the specific speed of the impeller using the European units system.

**Ns**

This property reports the specific speed of the impeller using the US units system.

**Nss**

This property reports the non-dimensional suction specific speed of the impeller.
**NumVanes**
This property specifies the number of impeller vanes.

**Type** \( \text{int} \)

**Read Only** No

**OmegaS**
This property reports the non-dimensional specific speed of the impeller.

**Type** \( \text{float} \)

**Read Only** Yes

**PowShaft**
This property reports the shaft power of the impeller.

**Type** \( \text{Quantity} \)

**Read Only** Yes

**PumpEff**
This property reports the overall efficiency of the impeller. This is the product of the hydraulic, volumetric and mechanical efficiencies.

**Type** \( \text{float} \)

**Read Only** Yes

**PumpEffUser**
This property specifies the impeller overall efficiency. This is valid when the EfficiencyOption is set to 'Hydraulic', 'Volumetric' or 'Mechanical'.

**Type** \( \text{float} \)

**Read Only** No

**R3**
This property reports the volute base-circle radius. This is defined as the distance from the centreline to the volute tongue.

**Type** \( \text{Quantity} \)

**Read Only** Yes

**Rake**
This property reports the lean angle at the impeller trailing edge (rake angle). Note that although this is also specified as an input property, the process to achieve the rake angle is iterative and may not
always be achievable. In this situation there will be a difference between this value and that specified by the input property.

**RakeUser**

This property specifies the blade lean angle at the impeller trailing edge (rake angle)

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Rho**

This property specifies the density of the working fluid.

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**RMajor**

This property reports a list of the major radii of the elliptic volute cross sections. This is valid when VoluteStyleOpt is 'Elliptic'.

<table>
<thead>
<tr>
<th>Type</th>
<th>List&lt;Quantity&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**RMinor**

This property reports a list of the minor radii of the elliptic volute cross sections. This is valid when VoluteStyleOpt is 'Elliptic'.

<table>
<thead>
<tr>
<th>Type</th>
<th>List&lt;Quantity&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**ShaftDiamRatio**

This property specifies the shaft diameter ratio. This is defined as the ratio of the hub diameter to the shaft diameter. It is used to determine the hub diameter from the shaft diameter and size of the impeller fittings used to fix the impeller to the shaft.

<table>
<thead>
<tr>
<th>Type</th>
<th>float</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
</tbody>
</table>

**ShrBeta1Opt**

This property specifies how the impeller leading edge blade angle at the shroud is set. With this option set to 'Incidence' the blade angle is calculated from the specified incidence, otherwise the blade angle is specified directly.
Available options:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incidence</td>
<td>User</td>
</tr>
<tr>
<td><strong>SlipRatio</strong></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Read Only</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Speed</strong></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Quantity</td>
</tr>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
<tr>
<td><strong>Theta2</strong></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Quantity</td>
</tr>
<tr>
<td>Read Only</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>ThetaCR</strong></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Quantity</td>
</tr>
<tr>
<td>Read Only</td>
<td>No</td>
</tr>
<tr>
<td><strong>ThetaDiff</strong></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Quantity</td>
</tr>
<tr>
<td>Read Only</td>
<td>Yes</td>
</tr>
</tbody>
</table>
**Thk**

This property reports the impeller vane thickness.

**Type**  
Quantity

**Read Only**  
Yes

**ThkRatio**

This property specifies the ratio of the impeller vane thickness to the tip diameter. It is used in order to specify the impeller vane thickness in a non-dimensional manner.

**Type**  
float

**Read Only**  
No

**TongueClear**

This property reports the volute tongue clearance. This is defined as the volute base-circle radius minus the impeller tip radius.

**Type**  
Quantity

**Read Only**  
Yes

**TongueThk**

This property reports the thickness of the volute tongue i.e. the tongue diameter at the cutwater.

**Type**  
Quantity

**Read Only**  
Yes

**U1**

This property reports the blade speed at the impeller leading edge meanline section.

**Type**  
Quantity

**Read Only**  
Yes

**U1Hub**

This property reports the blade speed at the impeller leading edge hub section.

**Type**  
Quantity

**Read Only**  
Yes

**U1Shr**

This property reports the blade speed at the impeller leading edge shroud section.

**Type**  
Quantity
**U2**

This property reports the blade speed at the impeller trailing edge (tip speed).

**Type** Quantity

**UserDiamDiff**

This property specifies that the volute diffuser exit diameter is to be defined by the user, rather than calculated automatically.

**Type** bool

**UserLengthDiff**

This property specifies that the volute diffuser axial length is to be defined by the user, rather than calculated automatically.

**Type** bool

**VolA**

This property reports a list of the volute cross sectional areas from cutwater to throat.

**Type** List<Quantity>

**VolEff**

This property reports the volumetric efficiency of the impeller.

**Type** float

**VolEffUser**

This property specifies the impeller volumetric efficiency. This is valid when the EfficiencyOption is set to 'Hydraulic', 'Mechanical' or 'Pump'.

**Type** float

**VolFlow**

This property specifies the design point volume flow rate delivered by the pump.
**Type**  
**Quantity**

**VolHeight**

This property reports a list of the height of the rectangular volute cross sections. This is valid when VoluteStyleOpt is 'Rectangular'.

**Type**  
List<Quantity>

**Read Only**  
Yes

**VolR**

This property reports a list of the radii of the centroids of the volute cross sections.

**Type**  
List<Quantity>

**Read Only**  
Yes

**VolRouter**

This property reports a list of the outer radii of the volute cross sections.

**Type**  
List<Quantity>

**Read Only**  
Yes

**VoluteStyleOpt**

This property specifies the volute cross section shape.

Available options:

- Elliptic
- Rectangular

**Type**  
VoluteType

**Read Only**  
No

**VolWidth**

This property reports a list of the width of the rectangular volute cross sections. This is valid when VoluteStyleOpt is 'Rectangular'.

**Type**  
List<Quantity>

**Read Only**  
Yes

**W1**

This property reports the relative flow velocity at the impeller leading edge meanline section.

**Type**  
Quantity
**Read Only**  Yes

**W1Hub**
This property reports the relative flow velocity at the impeller leading edge hub section.

**Type**  Quantity

**Read Only**  Yes

**W1Shr**
This property reports the relative flow velocity at the impeller leading edge shroud section.

**Type**  Quantity

**Read Only**  Yes

**W2**
This property reports the relative flow velocity at the impeller trailing edge.

**Type**  Quantity

**Read Only**  Yes

---

**Vista RTD**
This container holds Analysis data for an instance of Vista RTD.

**Methods**

**CreateBladeDesign**
This command class creates a new BladeGen model. An up-to-date BladeGen cell appears on the project schematic containing the new model.

**Optional Arguments**

**Beta**  Option to use the beta BladeGen template

  **Type**  bool

  **Default Value**  False

**CreateGeometry**
This command class creates a new BladeEditor model. An up-to-date Geometry cell appears on the project schematic containing the new model.
**CreateThroughflow**

This command class creates a new throughput system. The system, comprising Geometry, Setup, Solution and Results cells, is created on the project schematic and is updated performing the throughput analysis automatically.

**Optional Arguments**

**UseBladegen**
Indicates whether to use a Bladegen cell or a BladeEditor(Geometry) cell

<table>
<thead>
<tr>
<th>Type</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>False</td>
</tr>
</tbody>
</table>

**Edit**

This command class launches the Vista popup GUI.

**GetVistaRTDBladeDesignProperties**

This query takes a container reference and returns the Data Entity which contains user settings and properties for the VistaRTD Blade Design container.

- **Return**: Reference to the requested Data Entity
  - **Type**: DataReference

**ImportBladeGen**

This command imports Vista data into the Blade Design cell from an existing BladeGen file. If no appropriate Vista data is found in the specified BladeGen file, an error message is generated.

```python
template1 = GetTemplate(TemplateName="VistaCPD")
system1 = template1.CreateSystem()
bladeDesign1 = system1.GetContainer(ComponentName="Blade Design")
bladeDesign1.Import(FilePath="myfilepath/pump.bgd")
```

**Required Arguments**

**FileName**
Name, and path, of the BladeGen file to be imported.

- **Type**: string

**Data Entities**

**VistaRTDBladeDesign**

This data entity provides access to the properties which define the VistaRTD project. This includes both the input and the results properties.
Properties

AFR
This property specifies the air/fuel ratio of the working fluid. This is only available when using the AFR option for GasProps.

Type float
Read Only No

Alpha2
This property reports the absolute flow angle at the impeller inlet.

Type Quantity
Read Only Yes

Alpha2User
This property specifies the absolute flow angle at the impeller leading edge. Note that this is not available when the InletOption is set to 'Calculated'.

Type Quantity
Read Only No

Alpha3
This property reports the absolute flow angle at the impeller exit station.

Type Quantity
Read Only Yes

Alpha3User
This property specifies the absolute flow angle at the impeller trailing edge.

Type Quantity
Read Only No

Beta2
This property reports the relative flow angle at the impeller inlet.

Type Quantity
Read Only Yes

Beta2User
This property specifies the relative flow angle at the impeller leading edge. Note that this is not available when the InletOption is set to 'Calculated'.

Release 15.0 - © SAS IP Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
<th>Read Only</th>
<th>Beta3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>This property reports the meanline section relative flow angle at the impeller exit station.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
<th>Read Only</th>
<th>Beta3hub</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>This property reports the hub section relative flow angle at the impeller exit station.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
<th>Read Only</th>
<th>Beta3shroud</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>This property reports the shroud section relative flow angle at the impeller exit station.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Quantity</th>
<th>Read Only</th>
<th>Beta3User</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>This property specifies the relative flow angle at the impeller trailing edge.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>BMunitsType</th>
<th>Read Only</th>
<th>BMunitsOption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No</td>
<td>This property specifies the units used when creating a new BladeGen/BladeEditor model. Note that this is independent of the units used in the VistaRTD popup GUI.</td>
</tr>
</tbody>
</table>

Available options:

- mm
- cm
- inches
- ft
- m
**CalcInletAlphaOption**

When calculating the InletAngle from the NozzleArea there are 2 possible solutions, subsonic and supersonic. This property specifies whether to use a subsonic (LowSpeed) inlet Mach number, or a supersonic (HighSpeed) inlet Mach number.

Available options:

- LowSpeed
- HighSpeed

**ClearanceOption**

This property specifies impeller tip clearance is specified. 'Ratio' indicates that the tip clearance is specified as a fraction of the tip width 'User' specifies that the clearance will be defined directly by the user.

Available options:

- Ratio
- User

**ClearLoss**

This property reports the proportion of energy loss attributed to the clearances between rotating and stationary components. Note this is only available when using the 'Correlation' for the efficiency calculation method, EtaOpt.

**ClearRatioUser**

This property specifies the ratio of the impeller tip clearance to the tip width. Note that this is only available when ClearanceOption is set to 'Ratio'.

**ClearUser**

This property specifies the value of the impeller tip clearance. Note that this is only available when ClearanceOption is set to 'User'.

**Type**

- CalcType
- ClearanceType
- float
- float
- Quantity
**CorrelationOption**

This property specifies the correlation used to calculate the stage efficiency. Note that this is only available when the EtaOption is set to 'Correlation'.

Available options:

- Suhrmann
- Baines

**Type**  
**EtaCorrType**

**Read Only**  
No

**CpMean**

This property reports the average specific heat capacity at constant pressure for the turbine stage.

**Type**  
**Quantity**

**Read Only**  
Yes

**CpUser**

This property specifies the specific heat capacity at constant pressure, Cp, of the working fluid. This is only available when using the Fixed option for GasProps.

**Type**  
**Quantity**

**Read Only**  
No

**D2**

This property reports the impeller inlet diameter.

**Type**  
**Quantity**

**Read Only**  
Yes

**D3hub**

This property reports the impeller exit hub diameter.

**Type**  
**Quantity**

**Read Only**  
Yes

**D3shroud**

This property reports the impeller exit shroud diameter.

**Type**  
**Quantity**

**Read Only**  
Yes
**DiameterRatio**

This property reports the ratio of the impeller inlet diameter to the meanline exit diameter.

- **Type**: float
- **Read Only**: Yes

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**EtaImpTS**

This property reports the total-to-static impeller efficiency.

- **Type**: float
- **Read Only**: Yes

**EtaImpTT**

This property reports the total-to-total impeller efficiency.

- **Type**: float
- **Read Only**: Yes

**EtaNozzle**

This property specifies the value of the total-to-static nozzle efficiency. The term 'nozzle' here refers to the geometry upstream of the impeller used to control the inlet flow angle. This may be either bladed or unbladed, eg. a volute. A specified nozzle efficiency of 1.0 implies no pressure loss across the nozzle and consequently the nozzle is neglected from the calculation.

- **Type**: float
- **Read Only**: No

**EtaOption**

This property specifies whether to use a correlation to automatically calculate the turbine stage efficiency, or to use a user-defined efficiency.

Available options:

- User
- Correlation

```python
template1 = GetTemplate(TemplateName="VistaRTD")
system1 = template1.CreateSystem()
```
bladeDesign1 = system1.GetContainer(ComponentName="Blade Design")

vistaRTDProperties1 = bladeDesign1.GetVistaRTDBladeDesignProperties()

vistaRTDProperties1.EtaOption = "Correlation"

**Type**  
**EtaType**

**Read Only**  
No

**EtaStageTS**

This property reports the total-to-static stage efficiency.

**Type**  
float

**Read Only**  
Yes

**EtaStageTT**

This property reports the total-to-total stage efficiency.

**Type**  
float

**Read Only**  
Yes

**EtaUser**

This property specifies the value of the total-to-total turbine stage efficiency. Note that this entry is only available when the EtaOption is set to 'User'.

**Type**  
float

**Read Only**  
No

**ExitAngle**

This property specifies the flow angle at the impeller exit. This will either be an absolute or relative value depending on the 'ExitOption'.

**Type**  
Quantity

**Read Only**  
No

**ExitLoss**

This property reports the proportion of energy loss attributed to the exhaust. Note this is only available when using the 'Correlation' for the efficiency calculation method, EtaOpt.

**Type**  
float

**Read Only**  
Yes

**ExitOption**

This property specifies the ExitAngle type.

Available options:
Absolute
Relative

**Type**  
ExitAngleType

**Read Only**  
No

**ExpRatio**

This property specifies the design point total-to-total expansion ratio, P01/P03, for the turbine stage.

**Type**  
float

**Read Only**  
No

**ExpRatioTs**

This property reports the total to static expansion ratio for the turbine stage (P01/P3).

**Type**  
float

**Read Only**  
Yes

**FricLoss**

This property reports the proportion of energy loss attributed to friction. Note this is only available when using the 'Correlation' for the efficiency calculation method, EtaOpt.

**Type**  
float

**Read Only**  
Yes

**GammaMean**

This property reports the average ratio of specific heats (gamma) for the turbine stage.

**Type**  
float

**Read Only**  
Yes

**GasProps**

This property specifies the method used to calculate the properties of the working fluid.

Available options:

- Air
- AFR
- Fixed

```python
template1 = GetTemplate(TemplateName="VistaRTD")
system1 = template1.CreateSystem()
bladeDesign1 = system1.GetContainer(ComponentName="Blade Design")
vistaRTDProperties1 = bladeDesign1.GetVistaRTDBladeDesignProperties()
vistaRTDProperties1.GasProps = "AFR"
```

**Type**  
GasPropType
**HubRatio**

This property specifies the ratio of the impeller exit radius at the hub to the impeller inlet radius (tip radius). This enables the hub exit radius to be controlled in a non-dimensional way.

**Type** float

**ImpellerLength**

This property reports the ratio of the impeller axial length to the impeller tip diameter.

**Type** float

**ImpellerLengthOption**

This property specifies whether the impeller length ratio is calculated automatically, or specified by the user.

Available options:

- Automatic
- User

**Type** ImpellerLengthType

**ImpellerLengthUser**

This property specifies the ratio of the impeller axial length to the impeller tip diameter. This enables the impeller axial length to be controlled in a non-dimensional way.

**Type** float

**ImpellerLengthUserOpt**

This property specifies whether the impeller axial length ratio will be defined by the user.

**Type** bool

**ImpellerNumber**

This property specifies the number of impeller vanes.

**Type** int
**ImpellerThickness**

This property specifies the average vane thicknesses at the impeller exit.

- **Type**: Quantity
- **Read Only**: No

**IncLoss**

This property reports the proportion of energy loss attributed to the incidence at the leading edge. Note this is only available when using the 'Correlation' for the efficiency calculation method, EtaOpt.

- **Type**: float
- **Read Only**: Yes

**InletAngle**

This property specifies the flow angle at the impeller inlet. This will either be specified as an absolute or relative value depending on the 'InletOption'. Note that this is not available when the InletOption is set to 'Calculated'.

- **Type**: Quantity
- **Read Only**: No

**InletOption**

This property specifies the InletAngle type. If this is set to 'Calculated' then the InletAngle is calculated from the NozzleArea. If, in addition, ZeroBetaInlet is set, then CalcInletAlphaOption must also be specified.

Available options:
- Absolute
- Relative
- Calculated

- **Type**: InletAngleType
- **Read Only**: No

**Loading**

This property reports the basic loading for the turbine stage (\(\text{delH}/U^2\)).

- **Type**: float
- **Read Only**: Yes

**LoadLoss**

This property reports the proportion of energy loss attributed to loading. Note this is only available when using the 'Correlation' for the efficiency calculation method, EtaOpt.
Type float
Read Only Yes

**Mach2**

This property reports the absolute Mach number at the impeller inlet.

Type float
Read Only Yes

**Mach3rms**

This property reports the meanline section absolute Mach number at the impeller exit station.

Type float
Read Only Yes

**MachRel2**

This property reports the relative Mach number at the impeller inlet.

Type float
Read Only Yes

**MachRel3shroud**

This property reports the shroud section relative Mach number at the impeller exit station.

Type float
Read Only Yes

**MassFlow**

This property specifies the design point mass flow rate passing through the turbine stage.

Type Quantity
Read Only No

**MrootToverP**

This property reports the characteristic flow function for the design point \((M \times \sqrt{T} / P)\).

Type float
Read Only Yes

**NozChkRatio**

This property reports the choke ratio for the nozzle.
**Type**  float

**Read Only**  Yes

**NozExitDiameter**

This property reports the nozzle exit diameter.

**Type**  Quantity

**Read Only**  Yes

**NozThtArea**

This property reports the throat area of the nozzle.

**Type**  Quantity

**Read Only**  Yes

**NozVlessRatio**

This property reports the ratio of the 'vaneless' nozzle area to the impeller inlet radius. Note that the nozzle area used here neglects the vane thickness, hence the 'vaneless' prefix.

**Type**  Quantity

**Read Only**  Yes

**NozzleArea**

This property specifies the flow area of the nozzle at the throat. This is used to determine the impeller inlet flow angle when the 'Calculated' option is used for the InletOption.

**Type**  Quantity

**Read Only**  No

**NozzleNumber**

This property specifies the number of nozzle vanes.

**Type**  int

**Read Only**  No

**NozzleThickness**

This property specifies the average vane thicknesses at the nozzle throat.

**Type**  Quantity

**Read Only**  No
**Ns**
This property reports the specific speed of the impeller.

**Type**  float
**Read Only**  Yes

**Omega**
This property specifies the design point rotational speed of the impeller.

**Type**  Quantity
**Read Only**  No

**OvrChkRatio**
This property reports the overall choke ratio (mass flow/choke flow) for the turbine stage.

**Type**  float
**Read Only**  Yes

**P02**
This property reports the total pressure at impeller inlet.

**Type**  Quantity
**Read Only**  Yes

**P03**
This property reports the total pressure at impeller exit.

**Type**  Quantity
**Read Only**  Yes

**P2**
This property reports the static pressure at impeller inlet.

**Type**  Quantity
**Read Only**  Yes

**P3**
This property reports the static pressure at impeller exit.

**Type**  Quantity
**Read Only**  Yes
**Power**

This property reports the aerodynamic power generated by the turbine, neglecting mechanical losses.

**Type** Quantity

**Read Only** Yes

**Reaction**

This property reports the degree of reaction for the turbine stage, \((T2 - T3)/(T01 - T03)\).

**Type** float

**Read Only** Yes

**RelVelRatShroud**

This property reports the ratio of the impeller exit relative velocity at the shroud (\(W3shr\)) to the impeller inlet relative velocity (\(W2\)).

**Type** float

**Read Only** Yes

**RUUser**

This property specifies the specific gas constant, \(R\), of the working fluid. This is only available when using the Fixed option for GasProps.

**Type** Quantity

**Read Only** No

**ShroudRatio**

This property specifies the ratio of the impeller exit radius at the shroud to the impeller inlet radius (tip radius). This enables the shroud exit radius to be controlled in a non-dimensional way.

**Type** float

**Read Only** No

**SpanwiseDistributionOption**

This property specifies the spanwise distribution used when exporting the impeller blade.

Available options:

- General
- Radial

**Type** SpanwiseDistributionType

**Read Only** No
**SpeedRatio**

This property specifies the blade speed ratio, \( \frac{U_2}{C_0} \), of the impeller, where \( U_2 \) is the impeller tip speed and \( C_0 \) is the spouting velocity. In this definition the spouting velocity is calculated from the total-to-total isentropic temperature drop.

**Type** float

**Read Only** No

**StagPressure**

This property specifies the design point stagnation pressure, \( P_{01} \), at the inlet to the turbine stage. Note that this is defined upstream of the nozzle guide vane ahead of the impeller.

**Type** Quantity

**Read Only** No

**StagTemp**

This property specifies the design point stagnation temperature, \( T_{01} \), at the inlet to the turbine stage. Note that this is defined upstream of the nozzle guide vane ahead of the impeller.

**Type** Quantity

**Read Only** No

**SurfaceFinish**

This property specifies the surface finish of the impeller. The surface roughness has a secondary effect on the calculated efficiency.

Available options:

   Machined
   Cast

**Type** RoughnessType

**Read Only** No

**T02**

This property reports the total temperature at impeller inlet.

**Type** Quantity

**Read Only** Yes

**T03**

This property reports the total temperature at impeller exit.

**Type** Quantity
Read Only  Yes

**T2**

This property reports the static temperature at impeller inlet.

Type  Quantity

Read Only  Yes

**T3**

This property reports the static temperature at impeller exit.

Type  Quantity

Read Only  Yes

**TipWidth**

This property reports the impeller inlet tip width.

Type  Quantity

Read Only  Yes

**TotalLoss**

This property reports the sum of the friction, loading, clearance and exit losses. It is equal to one minus the total-to-total stage efficiency. Note this is only available when using the 'Correlation' for the efficiency calculation method, EtaOpt.

Total losses

Type  float

Read Only  Yes

**U2**

This property reports the blade speed at the impeller inlet (tip speed)

Type  Quantity

Read Only  Yes

**U3shroud**

This property reports the shroud section blade speed (tip speed) at the impeller exit station.

Type  Quantity

Read Only  Yes
This property reports the absolute velocity at the impeller inlet.

**Type**  
Read Only  

**V3rms**

This property reports the meanline section absolute velocity at the impeller exit station.

**Type**  
Read Only  

**Vax3rms**

This property reports the meanline section axial velocity at the impeller exit station.

**Type**  
Read Only  

**VelRatio1**

This property reports the flow coefficient at impeller exit. This is the ratio of the average axial velocity at the impeller exit (Vax3) to the impeller tip speed (U2).

**Type**  
Read Only  

**VelRatio2**

This property reports the blade speed ratio, U2/C0, of the impeller, where U2 is the impeller tip speed and C0 is the spouting velocity. In this definition the spouting velocity is calculated from the total-to-static isentropic temperature drop.

**Type**  
Read Only  

**Vr2**

This property reports the radial velocity at the impeller inlet.

**Type**  
Read Only  

**Vw2**

This property reports the swirl (tangential) velocity at the impeller inlet.

**Type**
**Read Only** Yes

**Vw3rms**
This property reports the meanline section swirl velocity at the impeller exit station.

**Type** Quantity

**Read Only** Yes

**W2**
This property reports the relative velocity at the impeller inlet.

**Type** Quantity

**Read Only** Yes

**W3shroud**
This property reports the shroud section relative velocity at the impeller exit station.

**Type** Quantity

**Read Only** Yes

**ZeroBetaInlet**
Specifies whether or not the relative flow angle at the impeller inlet is zero. Note that a zero inlet angle is a 'special case' where the exit angles are calculated rather than specified.

**Type** bool

**Read Only** No

---

**Vista TF Setup**
This container holds Setup data for an instance of Vista TF.

**Methods**

**GetSetupEntity**
This query is used to retrieve the Vista TF Setup data entity for the specified container.

**Return** A reference to the requested data entity.

**Type** DataReference

**ImportAeroTemplate**
Imports throughflow aerodynamic data into the Vista TF Setup from an existing throughflow aero template (.aert) file.
**Required Arguments**

**FilePath**
Path of the file to be imported.

Type  string

**ImportControlTemplate**

Imports throughflow control data into the Vista TF Setup from an existing throughflow control template (.cont) file.

**Required Arguments**

**FilePath**
Path of the file to be imported.

Type  string

**ImportCorrelationsTemplate**

Imports throughflow correlations data into the Vista TF Setup from an existing throughflow correlations template (.cort) file.

**Required Arguments**

**FilePath**
Path of the file to be imported.

Type  string

**ImportCustomRealGas**

Imports custom real gas data into the Vista TF Setup from a real gas file (.rgp)

**Required Arguments**

**FilePath**
Path of the file to be imported.

Type  string

**ImportGeometry**

Imports throughflow geometry data into the Vista TF Setup from an existing throughflow geometry (.geo) file.

**Required Arguments**

**FilePath**
Path of the file to be imported.

Type  string

**Data Entities**

**VistaTFSetup**

This data entity represents a Vista TF project definition on a geometry.
Properties

Acentric

User input, acentric factor

Type float
Read Only No

ConditionsFromUpstream

Flag to choose whether to update operating conditions from upstream cell or not

Type bool
Read Only No

CpCoeff

User input: Cp expression coefficients

Type List<double>
Read Only No

CpGas

Specifies the gas specific heat at constant pressure when FluidOption = IdealGas.

Type Quantity
Read Only No

CpMax

User input: Cp expression, max temperature

Type float
Read Only No

CpMin

User input: Cp expression, min temperature

Type float
Read Only No

CriticalPressure

User input, critical pressure

Type Quantity
Read Only  No

**CriticalTemp**

User input, critical temperature

**Type**  Quantity

Read Only  No

**CriticalVol**

User input, critical volume

**Type**  Quantity

Read Only  No

**CustomRealGasFilename**

The filename for the specified custom real gas file. Only used where the custom real gas option is specified.

**Type**  string

Read Only  No

**CwFluid**

Specifies the liquid specific heat when FluidOption = Liquid.

**Type**  Quantity

Read Only  No

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

**Type**  string

Read Only  No

**DynamicViscosity**

Specifies the fluid dynamic viscosity or machine Reynolds number.

Available options:

A value less than 1 [N s m⁻²] (or equivalent value in other units) is interpreted as the dynamic viscosity. Note that the value must be greater than 0.0000001 [N s m⁻²].

A value of 0 causes Vista TF to calculate the dynamic viscosity from an inbuilt equation for dynamic viscosity based on Sutherland’s law and the Inlet Total Temperature. This works only for an ideal gas.
A value greater than 1 \([N \cdot s \cdot m^{-2}]\) (or equivalent value in other units) is interpreted as the Reynolds number, in which case Vista TF calculates the dynamic viscosity using this Reynolds number, the Reference Diameter, the Machine Rotational Speed, and the fluid density.

**Type** Quantity

**Read Only** No

**FlowOption**

Specifies the types of boundary conditions.

Available options:

- **MassFlow**
  - Specify the mass flow rate and the inlet total pressure and total temperature conditions.

- **PressureRatio**
  - Specify the estimated mass flow rate and the total to static pressure ratio on the mean streamline.

- **PressureDifference**
  - Specify the estimated mass flow rate and the total to static pressure difference on the mean streamline.

**Type** FlowType

**Read Only** No

**FluidOption**

Specifies the type of fluid.

Available options:

- **IdealGas**
  - For compressible flows with ideal gas properties.

- **RealGas**
  - For compressible flow with real gas properties

- **Liquid**
  - For incompressible flows.

**Type** FluidType

**Read Only** No

**GammaGas**

Specifies the gas specific heat ratio when FluidOption = IdealGas.

**Type** float

**Read Only** No

**InitialCmUref**

Specifies the initial guess for the meridional velocity divided by a characteristic velocity, where the latter is half the Reference Diameter multiplied by the Machine Rotational Speed. For more information, see
the description for cm_start in Specification of the Control Data File (*.con) in the Vista TF Reference Guide.

**Type**  
float

**Read Only**  
No

**InletPt**

Specifies the inlet total pressure.

**Type**  
Quantity

**Read Only**  
No

**InletSwirlAngle**

Specifies the inlet swirl angle, which is from the axial direction and positive in the direction of rotation.

**Type**  
Quantity

**Read Only**  
No

**InletTt**

Specifies the inlet total temperature.

**Type**  
Quantity

**Read Only**  
No

**MachineTypeOption**

Specifies the type of machine that will be analyzed. This property specifies which template files are used and which report is used for the results. This property is only available if the geometry data is imported rather than transferred from BladeEditor.

Available options:

- Pump
- AxialCompressor
- CentrifugalCompressor
- Fan
- AxialTurbine
- RadialTurbine
- HydraulicTurbine
- Other
- Unknown

**Type**  
MachineType

**Read Only**  
No
MassFlow
Specifies the mass flow rate when FlowOption = MassFlow or the estimated mass flow rate when FlowOption = PressureRatio or PressureDifference.

Type Quantity
Read Only No

MaxIterations
Specifies the number of calculating streamlines used in the solver.

Type int
Read Only No

NumBladeRows
Specifies the number of blade rows solved in the analysis. This property is only available if the geometry data is imported rather than transferred from BladeEditor, and it is only used by the report template for the Results.

Type int
Read Only No

NumStreamlines
Specifies the number of calculating streamlines used in the solver.

Type int
Read Only No

PolytropicEfficiency
Specifies the small scale polytropic efficiency for the machine, and is used to calculate the losses.

Type float
Read Only No

PressureDifference
Specifies the total to static pressure difference on the mean streamline when FlowOption = PressureDifference.

Type Quantity
Read Only No

PressureRatio
Specifies the total to static pressure ratio on the mean streamline when FlowOption = PressureRatio.
**RealGasOption**

Specifies which real gas is used

_Type_  
RealGas

**RefDiameter**

Specifies the reference diameter for the definition of the flow coefficient.

_Type_  
Quantity

**RGas**

User input, specific gas constant

_Type_  
Quantity

**RhoFluid**

Specifies the fluid density when FluidOption = Liquid.

_Type_  
Quantity

**RotationalDirection**

Specifies the positive direction of machine rotation about the Z-axis.

Available options:

- RightHanded
- LeftHanded

_Type_  
RotationType

**RotationalSpeed**

The specified machine rotational speed.

_Type_  
Quantity
Vista TF Solution
This container holds Solution data for an instance of Vista TF.

Data Entities

VistaTFSolution
This data entity has no user modifiable properties.

Properties

DisplayText
The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string
Read Only: No
Units

This container holds the project Unit Systems and unit settings.

Data Entities

Quantity

Holds the unit definition for a specific quantity within a unit system.

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type: string
Read Only: No

IsSuppressed

True if this quantity is suppressed in the unit system.

Type: bool
Read Only: No

QuantityName

The name of the quantity.

Type: string
Read Only: No

Unit

The unit for the quantity.

Type: string
Read Only: No
**UnitSystem**

Holds the definition of a unit system within the project.

**Properties**

**DisplayText**

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

- **Type**: string
- **Read Only**: No

**IsDefault**

True if this is the current default unit system.

- **Type**: bool
- **Read Only**: No

**IsProjectUnitSystem**

True if this is the current project unit system.

- **Type**: bool
- **Read Only**: No

**Quantities**

The set of quantity entities that define the unit system.

- **Type**: `DataReferenceSet`
- **Read Only**: Yes

**UnitSystemName**

The internal name of the unit system.

- **Type**: string
- **Read Only**: Yes

**Methods**

**Export**

Exports unit system information into a Units XML file.

**Required Arguments**

- **FilePath**: The full path and name of the file to be created.
Type  string
Namespaced Commands

Ansoft

This namespace holds top-level commands and queries related to Ansoft.

AnalyzeDesignPointsInDesktop

Sends all the input parameters associated with the supplied system argument for analysis in the associated Ansoft product and solves for them. The designs points solved for depend on the scope argument.

Required Arguments

DesignPointScope This defines the collection of design points that will be analyzed.

Type EDesignPointScope

System The system whose associated design (and project) is to be analyzed/solved.

Type DataReference

CloseAllEditors

Closes all open editors associated with Ansoft systems on the workbench schematic

EditSystem

Edits the design associated with the system in an instance of the editor. If the system does not have any associated design, a new design is created/initialized for the system and that new design is opened for edit.

Required Arguments

System The system whose design is to be edited.

Type DataReference

ExportDesignPointsToDesktop

Exports the values of all the input paramters of the supplied System for all the design points covered by the supplied scope. The resulting table of values will be exported to the selected DXSetup's table in desktop.

Required Arguments

DesignPointScope The collection of design points for which the input parameter values of the supplied system will be exported.
Namespaced Commands

Type   EDesignPointScope

System   The system whose parameters will be considered for export.

Type   DataReference

**ForceSolutionIntoUpdateRequiredState**

Force the specified container’s component into the "UpdateRequired" or Lightning blot state to allow the associated editor to update its solution. This is primarily needed to incorporate feedback provided from downstream connected systems in a pseudo two-way multiphysics simulation.

**Required Arguments**

**System**   The system whose solution status is to be modified.

**Optional Arguments**

**InvalidateAllDesignPoints**   Whether this should invalidate all design points or just the current one. Default is true: invalidate all design points.

*Type*   bool

*Default Value*   True

**GetDesign**

Given an Ansoft system, obtain a proxy (IDispatch dynamic object) to the script/COM object corresponding to the design associated with the system. This will be more-or-less equivalent to the following in Desktop "oDesktop.GetActiveProject().GetActiveDesign()"

**Return**   The IDispatch wrapper representing the design script/IDispatch object.

*Type*   Object

**Required Arguments**

**System**   The ansoft system for which we want the Design script/IDispatch object.

**Type**   DataReference

**GetDesktop**

Given an Ansoft system, obtain a proxy (IDispatch dynamic object) to the script/COM object corresponding to the Desktop scripting object design associated with the system. This will be more-or-less equivalent to the "oDesktop" scripting object in Desktop’s CommandWindow

**Return**   The IDispatch wrapper representing oDesktop script/IDispatch object.

*Type*   Object
**Required Arguments**

**System**  The ansoft system for which we want the Desktop script/IDispatch object.

  **Type**  DataReference

**ImportProjectFile**

Import an existing Ansoft project file into workbench. The return value is a dictionary the keys of which are the systems created for the imported project and the values are the names of said systems.

**Return**  If successfully imported, the return value is a Dictionary mapping the DataReference of the created system to the name of the design in the imported project.

  **Type**  Dictionary<DataReference, string>

**Required Arguments**

**FilePath**  Absolute path to the file to import.

  **Type**  string

**LaunchDesktop**

Launches an instance of the editor associated with the System in standalone mode (cannot open workbench owned Ansoft projects). While the system itself cannot be edited (or even loaded) into the opened editor, application defaults and options can be set.

**Required Arguments**

**System**  The system that is used to figure out which editor is to be launched.

  **Type**  DataReference

**RefreshCurrentDesignPoint**

Synchronize Desktop variable values to current design point

**Required Arguments**

**System**  The system from which all design/project variables values will be synchronized to corresponding ones in the editor design.

  **Type**  DataReference
Namespaced Commands

**EKM**

This namespace holds top-level commands and queries related to Engineering Knowledge Manager.

**AutoregisterSession**

Auto-registers this Workbench session with the Workbench Job Manager object in EKM. This command is used when EKM starts a Workbench job. The registration details include hostname and port.

**ConnectEkmsServer**

Connect Workbench to an EKM server. To connect to a server in batch mode, there is at least one connection's credential (userName plus password) is saved during WB GUI mode. The only required parameter is the server location. Input UserName will be used to compare with saved credential. If no UserName provided, any of the saved user names will be used.

**Required Arguments**

Location EKM Server location to be connected

**Type** string

**Optional Arguments**

**IndividualServer** Specify if it's an Individual Server

**Type** bool

**Default Value** False

**Port** Port number for the server, default is 8080

**Type** long

**UserName** Specify a user name to the EKM server if needed

**Type** string

**Workspace** Workspace you wish to connect to

**Type** string

**Default Value** Default

**Example**

The following example illustrates the connecting to a given EKM server with other optional parameters.

```eetimes
EKM.ConnectEkmsServer(Location="auswchiang64", // Required
                    Workspace="Default", // Optional
                    Port=8080, // Optional
                    IndividualServer=False, // This parameter will eventually obsolete since individual server is removed from EKM
                    UserName="root" // Optional
                    )
```
**GetChangesFromRepository**

Get changes of the project from EKM repository.

**Optional Arguments**

- **BackupProject**
  - Description: Backup current project
  - **Type**: bool
  - **Default Value**: False

- **CheckOut**
  - Description: Checkout flag if under version controlled project
  - **Type**: bool
  - **Default Value**: False

**Example**

The following example illustrates the obtaining of project changes from an EKM repository.

```
EKM.GetChangesFromRepository(BackupProject=False,
                            Checkout=False)
```

**GetSessionId**

A query for getting the unique id of current Workbench session.

**Return**

The property value.

- **Type**: string

**LaunchWebUi**

Launches EKM Desktop.

**Required Arguments**

- **ConnectionUrl**
  - Description: Url to open in a web browser
  - **Type**: string

**OpenFromRepository**

Open an archived project from EKM repository to Workbench. The opened project will also saved in your local disk with repository info saved in the local project. Once it's opened, user can modify and save the project locally, send local changes to repository, and also update local project from repository if the project is changed in repository.

**Required Arguments**

- **ProjectPath**
  - Description: Local project path for the repository project to be opened to
  - **Type**: string
Namespaced Commands

**RepositoryPath**  
Remote repository path to be downloaded from EKM repository

  **Type**  
  string

Optional Arguments

**CheckOut**  
Should the repository project should be checked out if applicable

  **Type**  
  bool

  **Default Value**  
  False

Example

The following example illustrates the opening of an archive from an EKM Repository to local disk.

```csharp
EKM.SaveToRepository(RepositoryPath=r"\path_to_EKM_repository\archived_project.wbpz", // Required
                     ProjectPath=r"\local_project_path", // Required
                     CheckOut=False // Optional, applicable only if the project is under version control
                     )
```

**RegisterSession**

Registers the current Workbench session with EKM so that it can be controlled through EKM Web interface.

Required Arguments

**ConnectionName**  
Name of the connection

  **Type**  
  string

**SaveToRepository**

Save archived project to EKM repository. The project must be saved before calling this command. A warning will popup if not saved.

Required Arguments

**RepositoryFolderPath**  
Remote repository directory path for the archived file to be uploaded in the repository

  **Type**  
  string

Optional Arguments

**CheckOut**  
Check out if placing under version control

  **Type**  
  bool

  **Default Value**  
  False

**Description**  
Whether to include files imported to the project from external locations.

  **Type**  
  string
GetExclusiveControl  
Access Control  
Type  
bool  
Default Value  
False  

IncludeExternalFiles  
Include result files  
Type  
bool  
Default Value  
True  

IncludeResultFiles  
Include result files  
Type  
bool  
Default Value  
True  

PlaceUnderVersionControl  
Access Control  
Type  
bool  
Default Value  
False  

Example  
The following example illustrates the saving of an archive to an EKM Repository. It will save the specified archived project to the specified EKM repository, placing it under source control, and checking it out to my local machine.

```csharp
EKM.SaveToRepository(RepositoryFolderPath=r"\path_to_EKM_repository",
IncludeResultFiles=True,
IncludeExternalFiles=True,
GetExclusiveControl=False,
PlaceUnderVersionControl=False,
CheckOut=False,
Description="My Sample EKM Project Save")
```

SendChangesToRepository  
Save archived project to EKM repository. The project must be saved before calling this command. A warning will popup if not saved.

Required Arguments  
RepositoryPath  
Remote repository directory path for the project to be sent into repository  
Type  
string  

Optional Arguments  
Checkin  
Checkin option if applicable  
Type  
bool  
Default Value  
False  

---  
Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.
Namespaced Commands

**CheckOut**
CheckOut option if applicable

- **Type:** bool
- **Default Value:** False

**Comments**
Comments

- **Type:** string

**IncludeExternalFiles**
Include result files

- **Type:** bool
- **Default Value:** True

**IncludeResultFiles**
Include result files

- **Type:** bool
- **Default Value:** True

**ReleaseExclusiveControl**
Release exclusive control after update if applicable

- **Type:** bool
- **Default Value:** False

**Example**
The following example illustrates the sending of project changes to an EKM repository. The specified archived project will be sent to the specified EKM Repository. The additional arguments specify, in order: (1) Whether a new EKM project version will be assigned (InExiting). (2) Source control behavior (CheckinOption: a value greater than zero will issue a checkin. A value of 2 will check out the project after a successful checkin.). (3) Whether to release the exclusive control (lock) upon successful project submission (ReleaseExclusiveControl).

```python
EKM.SendChangesToRepository(RepositoryPath=r'\path_to_my_EKM_repository',
InExiting=True,
CheckinOption=0,
ReleaseExclusiveControl=False,
Comments="These are my comments for my changes")
```

**UnregisterSession**

Unregisters this Workbench session by deleting all job object(s) created in EKM to control it.

**UploadToRepository**

Save archived project to EKM repository. The project must be saved before calling this command. A warning will popup if not saved.

**Required Arguments**

- **ArchiveFilePath**
  Local file path for the archived project file to be save to repository
Type string

RepositoryFolderPath Remote repository directory path for the archived file to be uploaded in the repository

Type string

Optional Arguments

CheckOut Check out if placing under version control

Type bool

Default Value False

Description Whether to include files imported to the project from external locations.

Type string

GetExclusiveControl Access Control

Type bool

Default Value False

PlaceUnderVersionControl Access Control

Type bool

Default Value False

Example

The following example illustrates the saving of the specified archive to the specified EKM repository and updates the timestamp in the specified EKM record file. The additional arguments specify, in order: (1) Whether to obtain exclusive control for the archive after successful EKM upload (GetExclusiveControl). (2) Whether to place the uploaded archive under version control (PlaceUnderVersionControl). (3) Whether to check out the archived project after uploaded to EKM and placing it under version control (Checkout).

```python
EKM.UploadToRepository(ArchiveFilePath=r"C:\Users\anyuser\path_to_my_wbpz_file.wbpz",
RepositoryFolderPath=r"\path_to_my_EKM_repository",
LocalRepositoryRecordFilePath=r"C:\Users\anyuser\path_to_local_repository_project",
GetExclusiveControl=True,
PlaceUnderVersionControl=True,
CheckOut=True,
Comments="These are my comments for this update.")
```

EngData

This namespace holds top-level commands and queries related to Engineering Data.

CreateLibrary

Creates a new Engineering Data library.
Return

The created Engineering Data library.

Type: DataContainerReference

Required Arguments

Name
The name of a new library.

Type: string

Optional Arguments

FilePath
The target path for a new library.

Type: string

OpenLibrary

Opens a library of engineering information so that it can be viewed and if permissions allow, edited.

Return
The DataContainerReference for the library that was opened.

Type: DataContainerReference

Required Arguments

Source
The source of the library.

Type: string

Example

This code shows how to open a library from the provided samples.

```python
installDir = r"C:\Program Files\ANSYS Inc\v121"
library1 = EngData.OpenLibrary(
    Name="General Materials",
    Source=installDir+r"\Addins\EngineeringData\Samples\General_Materials.xml")
```

Extensions

This namespace holds top-level commands and queries related to ACT.

InstallExtension

This command install an extension identified by its filename.

If this extension already exists, the user can force to install the extension by setting the variable "ForceInstall" to true.

Required Arguments

ExtensionFileName
ExtensionFileName argument containing the file name of the extension.

Type: string
Optional Arguments

**ForceInstall**
Indicates whether the extension will be installed even if this extension already exists.

- **Type** : bool
- **Default Value** : False

**Example**
To install an extension:

```c
Extensions.InstallExtension(ExtensionFileName=r"c:\my_extensions_repository\my_extension.wbex")
```

To install an extension and force installation if the extension already exists:

```c
Extensions.InstallExtension(ExtensionFileName=r"c:\my_extensions_repository\my_extension.wbex", ForceInstall=True)
```

**LoadExtension**
This command load an installed extension.

**Required Arguments**

- **Id** : Id argument containing the GUID of the extension.

- **Type** : string

**Optional Arguments**

- **Format** : Format argument containing the format of the extension (Scripted or Binary).

- **Type** : string

- **Version** : Version argument containing the version of the extension.

- **Type** : string

**Example**
To load an extension:

```c
Extensions.LoadExtension(Id="7FD4DE83-39D0-4252-859B-2393C67F8EC8")
Extensions.LoadExtension(Id="7FD4DE83-39D0-4252-859B-2393C67F8EC8", Version="1.0")
Extensions.LoadExtension(Id="7FD4DE83-39D0-4252-859B-2393C67F8EC8", Version="1.0", Format="Binary")
```

**UninstallExtension**
This command uninstall an extension.

**Required Arguments**

- **Id** : Id argument containing the GUID of the extension.

- **Type** : string
Optional Arguments

**ForceUninstall** Indicates whether the extension will be uninstalled silently.

- **Type** `bool`
- **Default Value** `False`

**Format** Format argument containing the format of the extension (Scripted or Binary).

- **Type** `string`

**Version** Version argument containing the version of the extension.

- **Type** `string`

**Example**

To uninstall an extension:

```markdown
Extensions.UninstallExtension(ExtensionFileName=r"c:\my_extensions_repository\my_extension.wbex")
```

UnloadExtension

This command unload a loaded extension.

**Required Arguments**

**Id** Id argument containing the GUID of the extension.

- **Type** `string`

**Optional Arguments**

- **Format** Format argument containing the format of the extension (Scripted or Binary).
  - **Type** `string`
- **Version** Version argument containing the version of the extension.
  - **Type** `string`

**Example**

To unload an extension:

```markdown
Extensions.UnloadExtension(Id="7FD4DE83-39D0-4252-859B-2393C67F8EC8")
Extensions.UnloadExtension(Id="7FD4DE83-39D0-4252-859B-2393C67F8EC8", Version="1.0")
Extensions.UnloadExtension(Id="7FD4DE83-39D0-4252-859B-2393C67F8EC8", Version="1.0", Format="Binary")
```
**Graphics**

This namespace holds top-level commands and queries related to Graphics.

**GetAxisContinuous**

A query to return an AxisContinuous data reference.

**Return**

A reference to the AxisContinuous data entity.

**Type** DataReference

**Required Arguments**

**Name** The data entity name of the AxisContinuous object to be found.

**Type** string

**GetAxisDiscrete**

A query to return an AxisDiscrete data reference.

**Return**

A reference to the AxisDiscrete data entity.

**Type** DataReference

**Required Arguments**

**Name** The data entity name of the AxisDiscrete object to be found.

**Type** string

**GetChartXY**

A query to return a ChartXY data reference.

**Return**

A reference to the ChartXY data entity.

**Type** DataReference

**Required Arguments**

**Name** The data entity name of the ChartXY object to be found.

**Type** string

**GetChartXYZ**

A query to return an ChartXYZ data reference.

**Return**

A reference to the ChartXYZ data entity.

**Type** DataReference
**Namespaced Commands**

**Required Arguments**

**Name**  The data entity name of the ChartXYZ object to be found.

  **Type**  string

**GetCorrelationMatrix**

A query to return a CorrelationMatrix data reference.

**Return**  A reference to the CorrelationMatrix data entity.

  **Type**  DataReference

**Required Arguments**

**Name**  The data entity name of the CorrelationMatrix object to be found.

  **Type**  string

**GetLegend**

A query to return a Legend data reference.

**Return**  A reference to the Legend data entity.

  **Type**  DataReference

**Required Arguments**

**Name**  The data entity name of the Legend object to be found.

  **Type**  string

**GetMultiAxisChart**

A query to return a MultiAxisChart data reference.

**Return**  A reference to the MultiAxisChart data entity.

  **Type**  DataReference

**Required Arguments**

**Name**  The data entity name of the MultiAxisChart object to be found.

  **Type**  string

**GetPieChart**

A query to return a PieChart data reference.

**Return**  A reference to the PieChart data entity.

  **Type**  DataReference
Required Arguments

Name  The data entity name of the PieChart object to be found.

    Type  string

GetRenderStyle

A query to return a RenderStyle data reference.

Return  A reference to the RenderStyle data entity.

    Type  DataReference

Required Arguments

Name  The data entity name of the RenderStyle object to be found.

    Type  string

GetVariable

A query to return a Variable data reference.

Return  A reference to the Variable data entity.

    Type  DataReference

Required Arguments

Name  The data entity name of the Variable object to be found.

    Type  string

GetVariableXY

A query to return a VariableXY data reference.

Return  A reference to the VariableXY data entity.

    Type  DataReference

Required Arguments

Name  The data entity name of the VariableXY object to be found.

    Type  string

GetVariableXYZ

A query to return a VariableXYZ data reference.

Return  A reference to the VariableXYZ data entity.

    Type  DataReference
Namespaced Commands

Required Arguments

Name  The data entity name of the VariableXYZ object to be found.

   Type  string

Mechanical

This namespace holds top-level commands and queries related to Mechanical.

ImportLegacyDatabase

Imports a legacy database given the filepath

Return  A reference to the LegacyDataBaseResumeData object that contains information about the imported database

   Type  DataReference

Required Arguments

FilePath  Path to legacy database to import.

   Type  string

Meshing

This namespace holds top-level commands and queries related to Meshing.

ImportLegacyDatabase

Imports legacy Meshing editor files or CFX mesh files to the Meshing editor from an existing .cmdb file.

Return  The data references created by the import process.

   Type  DataReference

Required Arguments

FilePath  The file path to the legacy database to be imported.

   Type  string
Parameters

This namespace holds top-level commands and queries related to Parameters, Design Points and DesignXploration.

ClearDesignPointsCache

Clears the Design Points Cache for Design Exploration features. DesignXplorer is using an internal cache of Design Points to reduce the number of Design Point update operations. This cache is available across all of the design exploration systems of a project. This command clears all the data contained in the cache.

Example

The following example shows how to clear the cache in a project.

    ClearDesignPointsCache()

CreateDesignPoint

A command to create a new design point that contains all the parameters, but with their values set to null.

The design point can be created initially as a "exported" design point, which will allow the files and associated databases to be reloaded at a later time to view the results of the design point update.

Return

A DataReference to the new design point entity.

Type

DataReference

Optional Arguments

Exported

Indicates that the newly created design point will be exported.

Type

bool

Default Value

False

Retained

Indicates that the newly created design point will be retained.

Type

bool

Default Value

False

Example

The following example illustrates proper CreateDesignPoint command invocation:

    newDP = Parameters.CreateDesignPoint(Exported=True, Retained=False)
**CreateParameter**

Creates a parameter, optionally associated with a data entity property. The expression and visible name for the parameters can be optionally specified.

**Return**

A DataReference to the new parameter entity.

**Optional Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>displayText</strong></td>
<td>The display text for the parameter.</td>
</tr>
<tr>
<td><strong>entity</strong></td>
<td>The data model entity that holds the property to be parameterized.</td>
</tr>
<tr>
<td><strong>expression</strong></td>
<td>The initial expression for the parameter.</td>
</tr>
<tr>
<td><strong>isDirectOutput</strong></td>
<td>Indicates if the parameter is a direct output parameter.</td>
</tr>
<tr>
<td><strong>isOutput</strong></td>
<td>Indicates if the parameter is an output parameter.</td>
</tr>
<tr>
<td><strong>propertyName</strong></td>
<td>The name of the data model property which the parameter is associated with.</td>
</tr>
</tbody>
</table>

**Example**

The following two examples illustrate input and output parameter creation

```csharp
myEntity = #Query to obtain your entity on to which parameters will be created.
inptParam1 = Parameters.CreateParameter(Entity=myEntity, PropertyName="Value", IsOutput=False, IsDirectOutput=False, Expression=None, DisplayText="My Input Parameter")

outpParam1 = Parameters.CreateParameter(Entity=myEntity, PropertyName="Value", IsOutput=True, IsDirectOutput=True, Expression=None, DisplayText="My Output Parameter")

exprParam1 = Parameters.CreateParameter(Entity=None, PropertyName=None, IsOutput=False, IsDirectOutput=False, Expression="cos(1)", DisplayText="My Expression Parameter")
```
exprParam2 = Parameters.CreateParameter(Entity=None, PropertyName=None, IsOutput=True, IsDirectOutput=False, Expression="sin(P1)", DisplayText="My Output Expression Parameter")

CreateParameterSummaryChart

Creates a multi-axis parallel coordinate chart based on the parameters supplied to the command.

chart1 = Parameters.CreateParameterSummaryChart(Parameters=[])  
-or-
parameter1 = Parameters.GetParameter(Name="P1")
parameter2 = Parameters.GetParameter(Name="P2")
parameter3 = Parameters.GetParameter(Name="P3")
chart1 = Parameters.CreateParameterSummaryChart(Parameters=[parameter1, parameter2, parameter3])

Return  A data reference that represents the created chart.

Type  DataReference

Required Arguments

Parameters  The parameters to be included in the parallel coordinate plot.

Type  DataReferenceSet

CreateParameterVsParameterChart

Creates a 2-dimensional (x,y) chart that can be used to compare two parameters.

Return  A data reference that represents the chart that is created.

Type  DataReference

Optional Arguments

XAxisBottom  A data reference for the parameter that represents the bottom x-axis.

Type  DataReference

XAxisTop  A data reference for the parameter that represents the top x-axis.

Type  DataReference

YAxisLeft  A data reference for the parameter that represents the left y-axis.

Type  DataReference

YAxisRight  A data reference for the parameter that represents the right y-axis.

Type  DataReference

Example

This example illustrates the creation of a chart comparing two parameters.
parameter1 = Parameters.GetParameter(Name="P1")
parameter2 = Parameters.GetParameter(Name="P2")
chart1 = Parameters.CreateParameterVsParameterChart(XAxisBottom=parameter1, YAxisLeft=parameter2)

**ExportAllDesignPointsData**

Export the data of all design points to a csv file.

**Required Arguments**

**FileName** The exported file name.

*Type*  string

**Optional Arguments**

**AppendMode** True to append to an existing csv file, False to overwrite it.

*Type*  bool

*Default Value*  False

**Example**

The following example shows how the user can export all the design points of the project.

Parameters.ExportAllDesignPointsData(FileName="designPoints.csv")

**ExportLink**

Export the link to design set to a csv file.

**Required Arguments**

**FileName** The exported file name.

*Type*  string

**Optional Arguments**

**AppendMode** True to append to an existing csv file, False to overwrite it.

*Type*  bool

*Default Value*  False

**Example**

The following example shows how the user can export the dictionary of linked point.

Parameters.ExportLinkCommand(FileName="Link.csv")
**GetAllDesignPoints**

Returns a set of all design points in the project.

**Return**
A data reference set containing all the design points.

**Type**  
`DataReferenceSet`

**GetAllSortedDesignPoints**

Returns a set of all design points, sorted by their update order in ascending, in the project.

**Return**
A data reference set containing all the sorted design points.

**Type**  
`DataReferenceSet`

**GetAllExportedDesignPoints**

Retrieves a set of all exported design points.

**Return**
A set of all design points.

**Type**  
`DataReferenceSet`

**Optional Arguments**

**IncludingBaseDesignPoint**
A flag to determine whether the query to also return the base design point, which is always retained.

**Type**  
`bool`

**Default Value**
True

**GetAllParameters**

Returns a set of all parameters in the project.

**Return**
A data reference set containing all the parameters.

**Type**  
`DataReferenceSet`

**GetAllRetainedDesignPoints**

Retrieves a set of all retained design points.

**Return**
A set of all design points.

**Type**  
`DataReferenceSet`

**Optional Arguments**

**IncludingBaseDesignPoint**
A flag to determine whether the query to also return the base design point, which is always retained.

**Type**  
`bool`
**GetDesignPoint**

Returns a design point based on the name of the design point.

**Return**
The design point of interest.

**Type**  
DataReference

**Required Arguments**

**Name**  
The name of the design point.

**Type**  
string

**GetParameter**

A query to return a reference for a parameter given a name and a scope.

**Return**
The parameter of interest.

**Type**  
DataReference

**Required Arguments**

**Name**  
The name of the parameter.

**Type**  
string

**Optional Arguments**

**Scope**  
The scope in which the parameter is located. If this parameter is omitted or is null then the returned parameter will be global.

**Type**  
DataReference

**GetParameterSummaryChart**

A query to return a reference for a parameter summary chart by name.

**Return**
The parameter chart of interest.

**Type**  
DataReference

**Required Arguments**

**Name**  
The name of the chart.

**Type**  
string

**GetParameterVsParameterChart**

A query to return a reference for a parameter vs. parameter chart by name.
Return The parameter chart of interest.

Type DataReference

Required Arguments

Name The name of the chart.

Type string

IsParameterInDesignPointUpToDate

Checks whether a parameter is up to date in a design point.

Return True if the parameter is up to date in the design point.

Type bool

Required Arguments

DesignPoint The design point data reference.

Type DataReference

Parameter The parameter data reference.

Type DataReference

IsParameterUpToDate

Indicates whether a parameter is up to date in a design point.

Return Return if the parameter is up to date in the design point.

Type bool

Required Arguments

Parameter Parameter

Type DataReference

Optional Arguments

DesignPoint Design point

Type DataReference

OptimizeUpdateOrder

Optimizes Update Order of Design Points to minimize the number of modifications between two consecutive Design Points
**SetParameter**

Sets the expression of the specified parameter in the base design point.

**Required Arguments**

**Expression**
The string with the expression for the parameter.

**Type** string

**Parameter**
The parameter data reference.

**Type** DataReference

**SetUpdateOrderByRow**

Sets a value for the update order property of all design points using a sorting method.

**Optional Arguments**

**SortBy**
The definition of the sorting.

**Type** List<string>

---

**Project**

This namespace holds top-level commands and queries related to the Project, File and Units.

**AbandonBackgroundTasks**

Abandon all pending background tasks from last update.

**AbandonTopLevelBackgroundDesignPointUpdate**

Abandon all top-level pending background DP updates if none is running.

**AddTableColumn**

Add a column of data to a table

**Return**
The index of the column which was newly added to the table

**Type** int

**Required Arguments**

**Name**
The String used to uniquely identify the column in this table. If the value is not unique an error will occur.

**Type** string

**PhysicalQuantity**
The physical quantity (e.g. Density, Pressure, etc.) used to verify the data has the correct physical quantity.
**Type**

**string**

**Table**

The DataReference of the table to add this column of data.

**Type**

**DataReference**

**Optional Arguments**

**Data**

A List of the data for the column being added to the table.

**Type**

**List<Quantity>**

**Index**

The Integer index of the location for the newly added column. The default is to add the column to the rightmost side of the table.

**Type**

**int**

**Example**

This example illustrates adding a column of data to a table.

```python
import clr
clr.AddReference("Ans.Core")
import Ansys.Core
tableData = []
tableData.add(Ansys.Core.Units.Quantity(1.0, "m"))
table1 = ...
table1.AddColumn(Name="MyNewColumnData", PhysicalQuantity="Length", Data=tableData, Index=3)
```

**AddTableRow**

Add a row of data to a table.

**Return**

The index of the first row which was newly added to the table

**Type**

**int**

**Required Arguments**

**Table**

The DataReference of the table to add this row of data..

**Type**

**DataReference**

**Optional Arguments**

**Data**

A List of the quantities for the row being added to the table

**Type**

**List<Quantity>**

**Index**

The Integer index of the location for the newly added row. The default is to add the row to the end of the table.

**Type**

**int**

**Example**

This example illustrates adding a row of data in a table.
import clr
clr.AddReference("Ans.Core")
import Ansys.Core

tableData = []
tableData.add(Ansys.Core.Units.Quantity("1"))
table1 = ...
table1.AddRow(Data=tableData, Index=3)

Archive

Creates a project archive based on current project files and contents. The project must be saved before an archive can be created.

Required Arguments

FilePath The full file path of the project archive to be created.

  Type  string

Optional Arguments

FailIfMissingFiles Set to true to force failure if there are any files to repair.
  Type  bool
  Default Value  False

IncludeExternalImportedFiles Whether to include files imported to the project from external locations.
  Type  bool
  Default Value  False

IncludeSkippedFiles Whether to include Results and Solution files in the archive.
  Type  bool
  Default Value  True

IncludeUserFiles Whether to include files in the project user_files directory.
  Type  bool
  Default Value  True

Example

This example creates a project archive from the currently loaded project. The full file path is specified and the argument will include items from the user_files folder, include results/solution files, and exclude external files imported into the project.

    Archive(FilePath="C:\Users\AnsUser\simpleStructural.zip",
            IncludeUserFiles=True,
            IncludeExternalImportedFiles=False,
            FailIfFilesNeedRepair=False)
**CleanSystem**

Clears generated data on all components in the given system(s).

**Required Arguments**

- **Systems**  
  A list of systems to be cleaned.
  
  **Type**  
  List<DataReference>

**ClearMessages**

Clears all messages from the message store.

**ConvertRepositoryFileToLocalFile**

Convert a repository file to local file

**Required Arguments**

- **File**  
  The reference to the file entity to be converted.
  
  **Type**  
  DataReference

**CopyFile**

Copies a file to a target directory. Either the SourceFile argument or the SourceFilePath argument must be specified.

**Return**  
A reference to the created file is returned in this argument.
  
  **Type**  
  DataReference

**Required Arguments**

- **DestinationDirectoryPath**  
  The full path to the destination directory.
  
  **Type**  
  string

- **Overwrite**  
  Specifies whether to overwrite an existing files of the same name. Note: A registered file may not be overwritten.
  
  **Type**  
  bool

**Optional Arguments**

- **SourceFile**  
  A reference to the file to be copied. A file reference can be obtained from commands and queries like RegisterFileCommand and GetRegisteredFileQuery.
  
  **Type**  
  DataReference

- **SourceFilePath**  
  A full path to the file to be copied.
  
  **Type**  
  string
Example
This example illustrates a path-based file copy procedure. The specified file will be copied to the specified destination directory path. The destination file will not be registered and the return value will be null.

```python
CopyFile(SourceFilePath=r"C:\Users\anyuser\myTextFile.txt",
         DestinationDirectoryPath=r"C:\Users\anyuser\newfolder",
         Overwrite=True)
```

This second example illustrates a reference-based file copy procedure. The specified file will be copied to the specified destination directory path. The destination file will also be registered and the new reference returned.

```python
file1 = GetRegisteredFile(FilePath=r"C:\path_to_file\file.txt")
file2 = CopyFile(SourceFile=file1,
                  DestinationDirectoryPath=r"C:\Users\anyuser\newfolder",
                  Overwrite=True)
```

**CreateCustomUnitSystem**

Creates a custom unit system based on predefined unit system (e.g., MKS, US Customary) or other custom unit system already defined.

**Return**
A reference to the created UnitSystem data entity.

**Type**  DataReference

**Required Arguments**

**BaseUnitSystemName**
The internal name of the unit system that is the basis for the newly created system.

**Type**  string

**UnitSystemDisplayName**
The displayed name of the new unit system.

**Type**  string

**UnitSystemName**
The internal name of the new unit system.

**Type**  string

**Example**
This example illustrates the creation of a custom unit system.

```python
myUnitSystem = CreateCustomUnitSystem(UnitSystemName="MyUnitSystem",
                                       UnitSystemDisplayName="My Custom Unit System",
                                       BaseUnitSystemName="SI")
```

**CreateTemplateFromProject**

Creates a Project Template from the current project and adds it to the list of Custom systems.

**Return**
A reference to the created CustomProjectTemplate entity.
Required Arguments

Name  The name of the created template.

Type  string

CreateUnitSystem

Creates the UnitSystem data entity for the named system.

Return  A data reference to the created UnitSystem entity.

Type  DataReference

Required Arguments

UnitSystemName  The internal name of the unit system to be created.

Type  string

DeleteFile

Deletes the specified file. Either the File argument or the FilePath argument must be specified.

Optional Arguments

BackUp  Specifies whether to back up the file before deletion. This optional argument's default value is true.

Type  bool

Default Value  True

DeletelfShared  Specifies whether a registered file should be deleted even if it is still in use. This optional argument's default value is true. If this argument's value is false, an exception will be thrown if a shared file is encountered. To avoid the exception, set ErrorIfShared to false.

Type  bool

Default Value  False

ErrorIfShared  Specifies whether to throw an exception when trying to delete a shared file if DeletelfShared is set to false.

Type  bool

File  A reference to the file to be deleted. A file reference can be obtained from commands and queries like RegisterFileCommand and GetRegisteredFileQuery.

Type  DataReference

Example

The following example illustrates a simple file deletion; file is not deleted if it is registered.
DeleteFile(FilePath=r"C:\Users\anyuser\path-to-file.extension")

The next example illustrates forced file deletion.

DeleteFile(FilePath=r"C:\Users\anyuser\path-to-file.extension", DeleteIfShared=True)

The next example illustrates a deletion of a registered file via a file reference. The file will be deleted if even if it is still in use. Note that the file will be backed up.

fileRef = GetRegisteredFile(FilePath=r"C:\Users\anyuser\path-to-file.extension")
DeleteFile(File=fileRef, DeleteIfShared=True,
          BackUp=True)

The final example illustrates a deletion of a registered file via a file reference without a forced deletion. If the file is shared, an error will not occur, and the file reference count will be decremented.

fileRef = GetRegisteredFile(FilePath=r"C:\Users\anyuser\path-to-file.extension")
DeleteFile(File=fileRef, ErrorIfShared=False)

**DeleteTableColumn**

Deletes a column from a table.

**Required Arguments**

**Index**  
The column index to delete.

  **Type**  
  Object

**Table**  
The table from which to delete a column.

  **Type**  
  DataReference

**DeleteTableRow**

Deletes a row from the table.

**Required Arguments**

**Index**  
The row index to delete.

  **Type**  
  int

**Table**  
The table from which to delete a row.

  **Type**  
  DataReference

**DeleteTabularDataRow**

Delete a row from a tabular data sheet.
Required Arguments

**Index**

Index of the row to delete.

**Type**

`int`

**TabularData**

Data Entity that can be bound to `ITabularDataWrite`

**Type**

`DataReference`

Optional Arguments

**SheetName**

Name of the sheet to access.

**Type**

`string`

**SheetQualifiers**

SheetQualifiers is used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of the Qualifier and its Value.

**Type**

`Dictionary<string, string>`

Example

The following example illustrates the deletion of a row from a tabular data sheet.

```csharp
# Create a new Engineering Data System and access Structural Steel

template1 = GetTemplate(TemplateName="EngData")
system1 = template1.CreateSystem()
engineeringData1 = system1.GetContainer(ComponentName="Engineering Data")
matl1 = engineeringData1.GetMaterial(Name="Structural Steel")

# Delete the first row in the Density property

matlProp1 = matl1.GetProperty(Name="Density")
matlProp1.DeleteTabularDataRow(Index = 0)

# Delete the first row in the Coefficient of Thermal Expansion property with optional SheetName and SheetQualifiers

matlProp2 = matl1.GetProperty(Name="Coefficient of Thermal Expansion")
matlProp2.DeleteTabularDataRow(
    SheetName="Coefficient of Thermal Expansion",
    SheetQualifiers={"Definition Method": "Secant", "Behavior": "Isotropic"},
    Index = 0)
```

**DeleteUnitSystem**

Deletes the named unit system.

**Required Arguments**

**UnitSystemName**

The name of the unit system to delete.

**Type**

`string`

**DownloadRepositoryFile**

Download the specified file from repository
Namespaced Commands

**Return**

Holds the reference to the Repository file as the output value.

*Type* DataReference

**Required Arguments**

**RepositoryFile**

The repository file info to be downloaded

*Type* RepositoryFileInfo

**ExportReport**

Writes out project report containing both framework- and addin-supplied content. The report output is XML in accordance to EKM schema. If the file path extension is html, the XML will be converted to an html output file.

**Required Arguments**

**FilePath**

The full destination file path for the reporting. If the extension is .xml, the report will be produced in XML format per EKM schema. If the extension is .html, the report will be produced using the EKM schema-certified XML and then translated in html.

*Type* string

**GetAbsoluteUserPathName**

Gets the absolute user path name for the given relative path name, based on the current setting of the user path root. See also GetUserPathRoot and SetUserPathRoot.

**Return**

The returned absolute user path name.

*Type* string

**Required Arguments**

**RelativePathName**

The relative path name.

*Type* string

**Example**

This example uses the SetUserPathRoot and GetAbsoluteUserPathName to read a project from two different user directories.

```plaintext
SetUserPathRoot(DirectoryPath = "C:/Users/myUser1/Projects")
Open(FilePath=GetAbsoluteUserPathName("proj1.wbpj")) # Read project from first location
SetUserPathRoot(UserPathRoot = "C:/Users/myUser2/Projects")
Open(FilePath=GetAbsoluteUserPathName("proj1.wbpj")) # Read project from second location
```

**GetAllFiles**

Gets the list of all files that have been registered with the project (even if external to the project directory) or exist within the project files directory.

**Return**

List of paths to all known project files.
GetAllSystems

Query to return the set of Data References to all System entities in the project.

Return Data Reference set of Systems

Type DataReferenceSet

GetComponentTemplate

Query to return the Data Reference to the Component Template of the given name.

Return Component Template Data Reference.

Type DataReference

Required Arguments

Name Full name of the Component Template.

Type string

GetCurrentRegisteredFiles

Gets the files registered with the project.

Return The set of data references to registered files.

Type DataReferenceSet

GetDesignPointUpdateSettings

Query to return the singleton DesignPointUpdateSettings.

Return DesignPointUpdateSettings Data Reference.

Type DataReference

GetFileReference

Gets the File Reference to a registered file based upon the file reference's name. Throws an Invalid Operation Exception if a matching File Reference is not found.

Return The data reference of the file whose name matches the supplied parameter.

Type DataReference

Required Arguments

Name The file reference name.

Type string
**GetFilesForDirectory**

Gets the set of data references to registered files within a directory.

*Return*  
The set of data references to files in the directory.

*Type*  
DataReferenceSet

**Required Arguments**

**DirectoryName**  
The full path to the directory.

*Type*  
string

**GetFileType**

Returns a reference to a FileType data entity, given the data entity name of the FileType object.

*Return*  
The data reference to the FileType data entity.

*Type*  
DataReference

**Required Arguments**

**Name**  
The data entity name of the FileType object.

*Type*  
string

**GetFrameworkBuildVersion**

Get the current framework build version.

*Return*  
Return the framework build version.

*Type*  
string

**GetLicenseNames**

Query to return license features ID and names.

*Return*  
Returns license feature IDs along with readable names.

*Type*  
Dictionary<string, string>

**GetMaxProjectPathLength**

Gets the max project path length created in WorkBench based on the project directory and project name.

*Return*  
The max project path length

*Type*  
int
Required Arguments

**ProjectDirectory**
The full path to the project directory.

*Type*  string

**ProjectName**
The project name

*Type*  string

**GetMessages**

Returns DataReferences for all stored messages, ordered by message publication date/time with most recent messages first.

*Return*  The list of StoredMessage data entities for current messages.

*Type*  DataReferenceSet

**GetNeededLicenses**

Query to return the needed licenses to update project.

*Return*  Needed licenses based on previous updates, with license feature ID as key, along with needed count.

*Type*  Dictionary<string, int>

**GetPersistedProjectVersion**

Given the path to a WB project, return the last-saved-in version. Note: The command will return '00' if a project is not supplied and the current session project has not been saved.

*Return*  The Project’s persisted version (e.g, 120, 121, 130, 140, 145, ...).

*Type*  string

Optional Arguments

**FilePath**
The Project File Path (*.wbpj).

*Type*  string

**GetProjectDirectory**

Gets the full path to the current project directory.

*Return*  The project directory full path.

*Type*  string

**GetProjectFile**

Gets the full path of the current project file (*.wbpj).
**Namespaced Commands**

**Return**
The full path of the current project file.

**Type** string

**GetProjectTemplate**
Query to return the Project Template of the given name.

**Return**
Project Template Data Reference.

**Type** DataReference

**Required Arguments**

**Name**  
Full name of the Project Template.

**Type** string

**GetProjectUnitSystem**
Gets the current unit system for the project.

**Return**
A reference to the UnitSystem data entity.

**Type** DataReference

**GetPropertyExpression**
Gets the expression defining a data entity property's value.

**Return**
Returns the expression that currently defines the property's value.

**Type** string

**Required Arguments**

**Entity**  
The entity to query.

**Type** DataReference

**Name**  
The property name or member path.

**Type** string

**GetQuantityUnitForUnitSystem**
Gets the unit for a quantity in the specified unit system.

**Return**
The current Unit for the quantity.

**Type** string

**Required Arguments**

**QuantityName**  
The name of the Quantity of interest.
**UnitSystemName**

The internal name of the unit system.

**GetRegisteredFile**

Gets the File Reference to a registered file based upon the file's path.

**Return**

The data reference of the file whose location points to the specified path. Null if no data reference is found.

**Required Arguments**

**FilePath**

The full path of the file.

**GetRegisteredFilesForDirectory**

Gets list of File References for a directory path.

**Return**

The list of data reference of files whose location root is DirectoryPath. Empty list if no data reference is found.

**Required Arguments**

**DirectoryPath**

The full path to the directory.

**GetSchematicSettings**

Query to return the singleton SchematicSettings.

**Return**

SchematicSettings Data Reference.

**GetSolveManagerQueues**

Query to return queues available from a solver manager host.

**Return**

List of queues, or null if the SolverManagerHost is not ready or invalid host.
Namespaced Commands

**Required Arguments**

**SolverManagerHost**
Name of the solve manager

  **Type**  string

**GetSystem**

Query to return the Data Reference to the system of the given name.

**Return**
System Data Reference

  **Type**  DataReference

**Required Arguments**

**Name**
Full name of the System

  **Type**  string

**GetTableData**

Gets the data from the table.

**Return**
A List of Strings or a List of List of Strings for the requested range.

  **Type**  List<Object>

**Required Arguments**

**Table**
The DataReference of the table to get data from.

  **Type**  DataReference

**Optional Arguments**

**RowRangeEnd**
Ending index of row to get data from.

  **Type**  int

  **Default Value**  -1

**RowRangeStart**
Starting index of row to get data from.

  **Type**  int

  **Default Value**  0

**Example**
This example illustrates data retrieval from a table.

```
table1 = ...
tableData = table1.GetData(RowRangeStart=0, RowRangeEnd=2)
```
GetTabularData

Returns the tabular data associated with the data entity.

Return

The returned data in scalar, list, or dictionary format.

Type Object

Required Arguments

TabularData

Data entity that has associated tabular data.

Type DataReference

Optional Arguments

AsDictionary If set to true, the data will be returned as a dictionary where the keys are variable names and the values are the data for each variable. If set to false, the data will be returned in scalar or list format without the variable names.

Type bool

Default Value False

ColumnMajor If set to true, the data will be returned in column-major order. If set to false, the data will be returned in row-major order.

Type bool

Default Value True

EndIndex The end index for requesting a subset of the data (zero-based).

Type int

Default Value -2147483647

SheetName Specifies the sheet name when the data contains multiple sheets.

Type string

SheetQualifiers Used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of qualifiers and values.

Type Dictionary<string, string>

StartIndex The start index for requesting a subset of the data (zero-based).

Type int

Default Value 0

Variables Names of the variables for which data is requested (string or list of strings).

Type Object
Example
In this example, all data is requested for the given tabular data entity.

```
tabData1.GetData()
```

In this example, all data is requested in row-major order.

```
tabData1.GetData(ColumnMajor=False)
```

In this example, all data is requested in dictionary format.

```
tabData1.GetData(AsDictionary=True)
```

In this example, data for variables Density and Temperature is requested in dictionary format.

```
tabData1.GetData(Variables=\"Density\", \"Temperature\", AsDictionary=True)
```

GetAllTeamcenterConnections
A query to return all Teamcenter geometry connections.

**Return**
A list of Teamcenter source strings.

**Type**  
List<string>

GetTemplate
Query to return a System Template reference, based on the template name and additional solver specification.

**Return**
Template Data Reference.

**Type**  
DataReference

**Required Arguments**

TemplateName  
Generic name of the Template, without any solver-specific designation.

**Type**  
string

**Optional Arguments**

Solver  
The name of the solver to qualify the specific template.

**Type**  
string

GetUnitsDisplaySettings
Gets the value of units display settings ("DisplayValuesAsDefined" or "DisplayValuesInProjectUnits").

**Return**
The current value of the Units display setting.
**GetUnitSystem**

Gets a reference to the UnitSystem entity of the given internal name.

**Return**

A reference to the UnitSystem data entity.

**Type** DataReference

**Required Arguments**

**UnitSystemName**

The internal name of the unit system.

**Type** string

**GetUnitSystemsNames**

Gets the internal names of all currently defined unit systems (Predefined and Custom Unit Systems).

**Return**

A list of the internal names of currently defined unit systems.

**Type** List<string>

**GetUserFilesDirectory**

Gets the current user_files directory.

**Return**

The full path to the user_files directory.

**Type** string

**GetUserPathRoot**

Gets the current user path root.

**Return**

The user path root.

**Type** string

**ImportFile**

Imports a file into the project. The result of the import varies, dependent upon the file type’s defined import behavior. Usually a corresponding system is created in the project view. Sometimes, the file is simply added to the files view. To be imported, a file’s type must be registered and the ImportFileCommandName property must be non-null. Use FileRepository.RegisterTypeTemplate or FileTypeTemplateRecord.ImportFileCommandName to specify the command to be invoked when importing a file of a given type.

**Required Arguments**

**FilePath**

The full path to the file to import.

**Type** string
Optional Arguments

FileType  The reference to the type of the file to import. If not specified, the type will be inferred from the file's extension.

  Type  DataReference

ImportJournalVariables

Imports any variables defined in the currently-recorded journal file into the scripting environment so they can be accessed in the command window.

ImportUnitSystem

Imports a unit system from a Units .xml file.

Return  A reference to the created UnitSystem data entity.

  Type  DataReference

Required Arguments

FilePath  The full path to the file to be imported.

  Type  string

Optional Arguments

UnitSystemName  The internal name of the unit system to be created.

  Type  string

IsProjectUpToDate

A query to determine if all systems and their components in the project are up to date.

Return  The return is True if the project is up to date.

  Type  bool

MergeRsmProject

Merging multiple project tasks solved from RSM design point updates into the current project. This command should *NOT* be used to merge any un-related projects. Solved projects may be either intermediate or final results.

Required Arguments

MergingProjects  The dictionary contains full file path of base project of the RSM project as key, and a dictionary of the full file path to the updated RSM project and its design points to be merged into the current project as value. The file paths can be either standard wbpz file or partial wppz file or wbpj file. If the project is a standard wbpz or partial wppz archive, it will be unpacked to a sub-directory (whose name is the same as archive file name) under the same directory as the wbpz directory first before merging.
Type  Dictionary<string, Dictionary<string, List<DataReference>>>

Optional Arguments

**DoRevertibleSave**
Whether do revertible save or not

_Type_  bool
_Default Value_  True

**KeepFilesInRsmProject**
Determines whether we should keep the project files from the (unarchived) RSM project after it is merged into the current project. Note that keeping the RSM project will significant slow down the project file merging performance and will require more disk space. If this option is false, it is up to the caller to backup the original project first. Also note that if the ProjectFilePath is a *.wbpz file, it will not be affected.

_Type_  bool
_Default Value_  True

**Overwrite**
If the ProjectFilePath is an archive file (*.wbpz), and if the same unpacked project already exists on the same directory, this flag determines whether we should overwrite it.

_Type_  bool

**SaveFirst**
Whether to save the project before merging.

_Type_  bool

**Example**
The following example illustrates how this command may be used to merge all Design Points from RSM-solved projects into the current project. Existing Rsm Project files are retained. Any existing archive with the same name in the destination project will be overwritted. The project will be first saved using a revertible save.

```
MergeRsmProject(MergingProjects={(r"Path_to_RSM_Project", {(r"Path_to_current_project", {Updated DesignPoints})})},
KeepFilesInRsmProject=True,
Overwrite=True,
SaveFirst=True,
DoRevertibleSave=True)
```

**MoveFile**
Moves a file to a new directory. Either the File argument or the FilePath argument must be specified.

**Required Arguments**

**NewDirectoryPath**
The full path to the destination directory.

_Type_  string
Optional Arguments

**BackUp** Specifies whether to back up the file before the move. This optional argument’s default value is true.

- **Type**: bool
- **Default Value**: True

**File** A reference to the file to be moved. A file reference can be obtained from commands and queries like RegisterFileCommand and GetRegisteredFileQuery.

- **Type**: DataReference

**FilePath** A full path to the file to be moved.

- **Type**: string

**Example**
This example illustrates a path-based file move procedure. The specified file will be moved to the specified destination directory path.

```
MoveFile(FilePath=\"C:\Users\anyuser\myTextFile.txt\",
        NewDirectoryPath=\"C:\Users\anyuser\newfolder\")
```

This second example illustrates a reference-based file move procedure. The specified file will be moved to the specified destination directory path.

```
file1 = GetRegisteredFile(FilePath=\"C:\path_to_file\file.txt\")
MoveFile(SourceFile=file1,
        NewDirectoryPath=\"C:\Users\anyuser\newfolder\")
```

This final example also illustrates a reference-based file move procedure. The specified file will be moved to the specified destination directory path. The file will not be backed up prior to the move.

```
file1 = GetRegisteredFile(FilePath=\"C:\path_to_file\file.txt\")
MoveFile(SourceFile=file1,
        NewDirectoryPath=\"C:\Users\anyuser\newfolder\",
        Backup=False)
```

**Open**
Opens a Workbench project from the specified .wbpj file.

**Required Arguments**

**FilePath** The full path to the project file to open.

- **Type**: string
**Refresh**

Refreshes the input data for the entire project by reading changed data from upstream sources. Does not perform any calculations or updates based on the new data.

**RegisterFile**

Registers the specified file with the project. Either the FilePath argument or the RepositoryFile argument must be specified.

**Return**

Holds the reference to the registered file as the output value.

**Type** DataReference

**Optional Arguments**

**FilePath**

A full path to the file to be registered.

**Type** string

**FileType**

The reference to the type of the file to register. If not specified, the type will be inferred from the file's extension.

**Type** DataReference

**Example**

This example illustrates a path-based file registration procedure. The specified file will be registered and the resulting file reference will be returned.

```python
file1 = RegisterFile(FilePath=r"C:\Users\anyuser\myTextFile.txt")
```

This second example illustrates a file type based file registration procedure. The specified file will be registered with the corresponding (supplied) file type and the resulting file reference will be returned.

```python
fileType1 = GetFileType(Name="MyFileType")
file1 = RegisterFile(FilePath=r"C:\Users\anyuser\myFile.abc", FileType=fileType1)
```

**RenameFile**

Renames a file. Either the File argument or the FilePath argument must be specified.

**Required Arguments**

**New-Name**

The new file name, excluding the directory path. To change the directory, see MoveFileCommand.

**Type** string

**Optional Arguments**

**BackUp**

Specifies whether to back up the file before renaming it. This optional argument's default value is true.
**RenameFile**

A reference to the file to be renamed. A file reference can be obtained from commands and queries like RegisterFileCommand and GetRegisteredFileQuery.

**Type**  
DataReference

**FilePath**  
A full path to the file to be renamed.

**Type**  
string

**Example**

This example illustrates a path-based file rename procedure. The specified file will be renamed with the specified new name.

```plaintext
RenameFile(FilePath=r"C:\Users\anyuser\myTextFile.txt",
       NewName="myTextFileRENAMEd.txt")
```

This second example illustrates a reference-based file rename procedure. The specified file reference will be renamed with the specified new name.

```plaintext
file1 = GetRegisteredFile(FilePath=r"C:\path_to_file\file.txt")
RenameFile(File=file1,
       NewName="fileRENAMEd.txt")
```

This final example also illustrates a reference-based file rename procedure. The specified file will be renamed with the specified new name. The file will not be backed up prior to the move.

```plaintext
file1 = GetRegisteredFile(FilePath=r"C:\path_to_file\file.txt")
RenameFile(File=file1,
       NewName="fileRENAMEd.txt",
       Backup=False)
```

**RenameUnitSystem**

Renames a unit system.

**Required Arguments**

**UnitSystemName**  
The name of the unit system to be changed.

**Type**  
string

**UnitSystemNewName**  
The new name of the unit system.

**Type**  
string

**Reset**

Resets Workbench to an empty project. Any unsaved changes in the current project are lost.
Resolve

A command used to resolve the data object to make it consistent with the rest of the data model

Required Arguments

Entity  The entity to resolve.

  Type  DataReference

ResumeBackgroundDesignPointUpdate

Resume given design point update session previously submitted for remote execution.

Return  Optional output which will be set immediately to the input session. It is provided for the convenience of the invoking code so that the same type of output can be returned from this vs. UADP command.

  Type  DataReference

Optional Arguments

Session  If not specified, top level background UADP sessions will be resumed.

  Type  DataReference

ResumeDesignPointUpdates

Resume design point updates suspended waiting for subtasks executed in background; or resume suspended parametric updates.

RunScript

Executes a Workbench script file from the specified location.

Required Arguments

FilePath  Full path to the script file to execute.

  Type  string

Save

Saves the current project to disk.

Optional Arguments

FilePath  The full path of the project file to be saved.

  Type  string

Overwrite  Whether to overwrite the project save location if it exists.

  Type  bool
SaveProjectArchiveToTeamcenter

Saves a project archive to Teamcenter. The project archive must be created before calling this function.

**Required Arguments**

- **ArchiveFileName**
  - **Type**: string
  - **Description**: Archive file name

- **TeamcenterUserName**
  - **Type**: string
  - **Description**: Teamcenter User Name

- **TeamcenterUserPassword**
  - **Type**: string
  - **Description**: Teamcenter User Password

**Optional Arguments**

- **DatasetDescription**
  - **Type**: string
  - **Description**: Item description

- **DatasetName**
  - **Type**: string
  - **Description**: Dataset name. ArchiveFileName will be considered as the Dataset name, if (ItemName and ItemRevision) or Dataset name is not specified

- **DeleteArchiveOnExit**
  - **Type**: bool
  - **Default Value**: False
  - **Description**: Deletes the archive file name on exiting the process

- **ItemName**
  - **Type**: string
  - **Description**: Teamcenter Item name

- **ItemRevision**
  - **Type**: string
  - **Description**: Teamcenter Item Revision

- **ItemRevisionUID**
  - **Type**: string
  - **Description**: Teamcenter Item Revision UID

- **ItemSequence**
  - **Type**: string
  - **Description**: Item sequence number

- **ItemUID**
  - **Type**: string
  - **Description**: Teamcenter Item UID

- **SSLCertificateFile**
  - **Type**: string
  - **Description**: Teamcenter SSL CA Certificate File for HTTPS connections
**TeamcenterDiscriminator**
Teamcenter Discriminator

**TeamcenterGroup**
Teamcenter Group

**TeamcenterLocale**
Teamcenter Locale. eg. en_US

**TeamcenterRole**
Teamcenter Role

**WaitForExitMillisecond**
Wait till teamcenter checkin is finished

**Default Value** 30000

**Example**
Saves a project archive to Teamcenter

```csharp
SaveProjectArchiveToTeamcenter(ArchiveFileName=r"C:\Users\AnsUser\simpleStructural.wbpz",
TeamcenterUserName="infodba",
TeamcenterUserPassword="infodba",
ItemName="ANS0048",
ItemRevision="001",
Description="Workbench2 Project Archive")
```

**SetDesignPointUpdateOption**
Set the design point updates run mode.

**Required Arguments**

**UpdateOption** Update option to be set.

**Type** JobRunMode

**SetProjectUnitSystem**
Sets the unit system for the current project.

**Return** A reference to the UnitSystem entity that is now the project unit system.

**Type** DataReference
Namespaced Commands

Required Arguments

**UnitSystemName**
The internal name of the unit system to be used in the project.

Type: string

**SetQuantityUnitForUnitSystem**

Sets the unit for a Quantity in the specified unit system. Changing Base or Common units will change derived units if appropriate. Note: Quantity unit for a predefined unit system cannot be changed. Note: Only consistent units (SI or US Customary) are allowed.

*Return*
A dictionary of the Quantity names and new units for all resulting quantity changes. This will be more than just the target quantity if a base unit is changed.

Type: Dictionary<string, string>

Required Arguments

**QuantityName**
The name of the Quantity to be changed.

Type: string

**Unit**
The new Unit for the quantity.

Type: string

**UnitSystemName**
The internal name of the unit system to be changed.

Type: string

**Example**
The following example illustrates the setting of a quantity unit in a given unit system.

```csharp
changedUnits = SetQuantityUnitForUnitSystem(UnitSystemName="myUnitSystem",
                                           QuantityName="Temperature",
                                           Unit="C")
```

**SetScriptVersion**

Sets the command API version used when executing a script.

Required Arguments

**Version**
The command API version required to run the script.

Type: string

**SetTableData**

Sets the data in the table.

Sets the data in the table.
Required Arguments

**Column**  
A string representation of index or string name of the column to begin setting data. The first column has an index of 0 (zero). The column string name is case sensitive.

  **Type**  
  Object

**Data**  
A List of Quantities which specifies the data to set at the given row and column of the table

  **Type**  
  List<Quantity>

**Row**  
The Integer index of the row of the table to begin setting data. The first row has an index of 0 (zero).

  **Type**  
  int

**Table**  
The DataReference of the table which will have data changed.

  **Type**  
  DataReference

Optional Arguments

**DataOrder**  
An enum of Row or Column which indicates if the Data is used in a row-major or column-major order. The default is Row.

  **Type**  
  DataOrder

  **Default Value**  
  Row

Example  
This example illustrates setting data in a table.

```python
import clr
clr.AddReference("Ans.Core")
import Ansys.Core
tableData = []
tableData.add(Ansys.Core.Units.Quantity(1.0, "m"))
table1 = ...
table1.SetData(Row=0, Column=0, Data=tableData)
```

SetTabularData

Set tabular data associated with the data entity.

Required Arguments

**TabularData**  
Data entity that has associated tabular data.

  **Type**  
  DataReference

Optional Arguments

**Data**  
Sets the data using a dictionary form. The keys are the variable names and the values are the data. The use of this argument is mutually exclusive with "Values" and "Variables".
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Specifies the starting location used to set the data (zero-based). A value of -1 indicates that the data should be appended to the existing data.</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>SheetName</td>
<td>Specifies the sheet name when the data contains multiple sheets.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>SheetQualifiers</td>
<td>Used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of qualifiers and values.</td>
<td>Dictionary&lt; string, string&gt;</td>
<td></td>
</tr>
<tr>
<td>Values</td>
<td>List of data values set in conjunction with the &quot;Variables&quot; parameter. This parameter and the &quot;Data&quot; parameter are mutually exclusive.</td>
<td>List&lt;List&lt;Object&gt;&gt;</td>
<td></td>
</tr>
<tr>
<td>Variables</td>
<td>Names of the variables for which data is being set. This parameter and the &quot;Data&quot; parameter are mutually exclusive.</td>
<td>List&lt;string&gt;</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

```python
# Create a new Engineering Data System and access Structural Steel

# template1 = GetTemplate(TemplateName="EngData")
# system1 = template1.CreateSystem()
# engineeringData1 = system1.GetContainer(ComponentName="Engineering Data")
# matl1 = engineeringData1.GetMaterial(Name="Structural Steel")
#
# Change the value of a simple single-valued property
#
# matlProp1 = matl1.GetProperty(Name="Density")
# matlProp1.SetData(
#     Variables="Density",
#     Values="8500 [kg m^-3]"
# )
#
# Set Temperature-dependent data for Elasticity based
# on lists of variables and values.
#
# temperature = ["400 [K]", "600 [K]", "800 [K]"]
# E = ["2e5 [MPa]", "1.9e5 [MPa]", "1.6e5 [MPa]"]
# matlProp2.SetData(
#     Variables = ["Temperature","Young's Modulus"],
#     Values = [temperature, E])
#
# Change the Temperature for the second table entry.
#
# matlProp2.SetData(
#     Index = 1,
#     Variables = "Temperature",
#     Values = "625 [K]"
# )
#
# Set a list for Poisson's Ratio starting at the second table entry.
#```
matlProp2.SetData(
    Index = 1,
    Variables = "Poisson's Ratio",
    Values = [0.3, 0.3])

# Set Temperature-dependent property data for the Coefficient of Thermal Expansion
# using a dictionary. The dictionary key is the Variable name,
# followed by the list of values for the variable.
# matlProp3 = matl1.GetProperty(Name="Coefficient of Thermal Expansion")
newData = {
    "Temperature": ["200 [F]", "400 [F]", "600 [F]", "800 [F]", "1000 [F]"],
    "Coefficient of Thermal Expansion": ["6.3e-6 [F^-1]", "7.0e-6 [F^-1]",
    "7.46e-6 [F^-1]", "7.8e-6 [F^-1]",
    "8.04e-6 [F^-1]"]
}
matlProp3.SetData(
    SheetName="Coefficient of Thermal Expansion",
    SheetQualifiers={"Definition Method": "Secant", "Behavior": "Isotropic"},
    Data = newData)

**SetTabularDataQualifier**

Changes the values of a specific qualifier in a data table.

**Required Arguments**

**Qualifier**

The Qualifier to Set.

*Type* string

**TabularData**

Data Entity that can be bound to ITabularDataWrite

*Type* DataReference

**Value**

The new value.

*Type* string

**Optional Arguments**

**SheetName**

The name of the tabular data sheet that contains the qualifier.

*Type* string

**SheetQualifiers**

SheetQualifiers can be used to pass in the qualifiers to select between multiple sheets with the same name. This is a dictionary of the Qualifier and its Value.

*Type* Dictionary<string, string>

**VariableName**

The name of the Variable that contains the qualifier to be changed.

*Type* string

**VariableQualifiers**

VariableQualifiers can be used to pass in the qualifiers to select between multiple variables with the same name. This is a dictionary of the Qualifier and its Value.

*Type* Dictionary<string, string>
Example
The following example changes the 'Derive From' setting within an Isotropic Elasticity material property to be "Bulk Modulus and Poisson's Ratio".

```python
mat1 = engineeringData1.GetMaterial(Name="Structural Steel")
mat1Prop1 = mat1.GetProperty(Name="Elasticity")
mat1Prop1.SetQualifier(
    SheetName="Elasticity",
    Qualifier="Derive from",
    Value="Bulk Modulus and Poisson's Ratio")
```

**SetUnitsDisplaySettings**

Sets units display settings.

**Required Arguments**

**DisplaySettings** The new value to be used for Unit display settings. Allowed values are:
- DisplayValuesAsDefined
- DisplayValuesInProjectUnits

**Type** string

**SetUserPathRoot**

Sets the current user path root. The root path facilitates portability of session journals by allowing relative rather than absolute paths to be recorded. During command journal replay, paths are reconstructed using the active user path root setting.

**Required Arguments**

**DirectoryPath** The new root directory path.

**Type** string

**Unarchive**

Restores a project from an archive to the specified location.

**Required Arguments**

**ArchivePath** The project archive file path to open.

**Type** string

**Overwrite** Whether the project should be overwritten if it already exists.

**Type** bool

**Optional Arguments**

**ProjectPath** The path to which the archived project will be uncompressed. If not specified, it will be extracted to a directory under "ProjectTemporaryFilesFolder" set by user preference. If the
"ProjectTemporaryFilesFolder" is not set, it will be extracted to a directory under system temp directory.

**Type** string

**Example**
This example illustrates an unarchive procedure. The specified archive will be un-archived to the specified project file path. If the project file path already exists, it will be overwritten since Overwrite=True.

```python
Unarchive(ArchivePath=r"C:\Users\anyuser\myProjectArchive.wbpz",
          ProjectFilePath=r"C:\Users\anyuser\myProject.wbpj",
          Overwrite=True)
```

**UpdateAllDesignPoints**

Updates all design points or a subset of them if specified.

**Return**
The reference to the DesignPointUpdateBackgroundSession object, if the command is executed via RSM.

**Type** DataReference

**Optional Arguments**

**CannotCompleteBehavior** Determines the action to take when a component cannot be updated but does not have an error, for example if it needs user intervention.

**Type** UpdateErrorBehavior

**Default Value** Continue

**DesignPoints**
The set of design points to update. If this parameter is not given, all out of date design points are updated.

**Type** List<DataReference>

**ErrorBehavior** Determines the action to take when an error is encountered.

**Type** UpdateErrorBehavior

**Default Value** Continue

**Parameters**
The set of parameters to update. If this parameter is not given, all parameters are updated.

**Type** List<DataReference>

**Example**
The following example illustrates a more verbose usage of UpdateAllDesignPoints, as general users will typically call just UpdateAllDesignPoints(). The following call processes two design points for all parameters (all when None is specified), skips the executing design point when observing an error, and stops if a design point update cannot be completed.

```python
dps = []
dp = Parameters.GetDesignPoint("1")
```
Namespaced Commands

dps.append(dp)
dp = Parameters.GetDesignPoint("2")
dps.append(dp)
UpdateAllDesignPoints(DesignPoints=dps,
    Parameters=None,
    ErrorBehavior="SkipDesignPoint",
    CannotCompleteBehavior="Stop")

**Update**

Updates all components in the project by refreshing all input data and performing local calculations to produce new output if the component is out of date.

**ValidateDesignPointUpdateSettings**

Validate DesignPointUpdateSettingsEntity initialization.
Data Types

AdaptKrigOutType

Enumeration for the Output Variable Combinations type when refining for the Kriging algorithm.

Possible Values

| AK_MAXOUT | Maximum Output |
| AK_ALLOUT | All Outputs |
| AK_SUMOUT | Sum of Outputs |

AnalysisType

3D/2D import option

Possible Values

| AnalysisType_3D | Import all 3D objects |
| AnalysisType_2D | Import only 2D objects (The model must be in the x-y plane.) |

AxesRangeModes

Enumeration of the Automatic Range modes for output parameters.

Possible Values

| OutputParameterMinMax | The range of the output parameter axis is determined from the minimum and maximum of the parameter (min-max search of DPs min-max) |
| ChartData | The range of the output parameter axis is determined by the min and max of the chart's data. |

Axis

The specification of allowed chart axes.
**Possible Values**

None  
X_Axis  
Y_Axis  
Z_Axis

**BladeLoftType**

Enumeration of the possible blade lofting directions.

**Possible Values**

Streamwise  
Spanwise

**BladeType**

Blade type for export to BladeGen

**Possible Values**

IGV  
Rotor  
OGV

**BladeType**

Blade type for export to BladeGen

**Possible Values**

IGV  
Rotor  
OGV

**BMunitsType**

BladeGen/BladeEditor units type

**Possible Values**

m  
cm  
mm  
inches  
ft

Create blade model in metres  
Create blade model in cm  
Create blade model in mm  
Create blade model in inches  
Create blade model in feet
**Possible Values**

- mm: Create blade model in mm
- inches: Create blade model in inches

**BodyGrouping**

The body grouping property used when importing the model.

**Possible Values**

- None: No body grouping action should be performed.
- Material: The model should be grouped by material IDs.
- ElementType: The model should be grouped by element type IDs.
- Thickness: The model should be grouped by thickness IDs.
- Components: The model should be grouped by component IDs.
- Unknown: The body grouping property is not set.

**bool**

This type represents a Boolean value. Valid values are 'True' or 'False'.

**CalcType**

Calculation from nozzle area type

**Possible Values**

- HighSpeed: High speed inlet velocity calculation
- LowSpeed: Low speed inlet velocity calculation

**CandidatesColoringMethods**

Enumeration of the Coloring methods used for Candidates chart.

**Possible Values**

- ColoringPerPointType: The color is used to distinguish the different types of candidate points (Starting Points, Candidate Points, etc).
- ColoringPerOutputNature: The color is used to distinguish the nature of the output values (Response Surface or Simulation).

**CasePrecision**

Precision of FLUENT Session.
**Possible Values**

Single
Double

**CCDTemplateType**

Enumeration for the Template Type for CCD algorithm.

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCD_STANDARD_TEMPLATE</td>
<td>Standard</td>
</tr>
<tr>
<td>CCD_ENHANCED_TEMPLATE</td>
<td>Enhanced</td>
</tr>
</tbody>
</table>

**CdfPlotType**

Enumeration of the available Cumulative Distribution Plot types.

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Uniform</td>
<td>Uniform</td>
</tr>
<tr>
<td>Triangular</td>
<td>Triangular</td>
</tr>
<tr>
<td>Gauss</td>
<td>Normal</td>
</tr>
<tr>
<td>LogNorm</td>
<td>Lognormal</td>
</tr>
<tr>
<td>Exponential</td>
<td>Exponential</td>
</tr>
<tr>
<td>Beta</td>
<td>Beta</td>
</tr>
<tr>
<td>Weibull</td>
<td>Weibull</td>
</tr>
</tbody>
</table>

**CentralCompositeDesignType**

Enumeration of the available design types for the Central Composite Design algorithm.

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCDTYPE_FACE_CENT</td>
<td>Face-Centered</td>
</tr>
<tr>
<td>CCDTYPE_ROT</td>
<td>Rotatable</td>
</tr>
<tr>
<td>CCDTYPE_VIF_OPT</td>
<td>VIF-Optimality</td>
</tr>
<tr>
<td>CCDTYPE_G_OPT</td>
<td>G-Optimality</td>
</tr>
<tr>
<td>CCDTYPE_AUTO</td>
<td>Auto Defined</td>
</tr>
</tbody>
</table>

**ChartAxes**

The possible chart axes.

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XAxis</td>
<td>X axis</td>
</tr>
<tr>
<td>YAxis</td>
<td>Y axis</td>
</tr>
<tr>
<td>ZAxis</td>
<td>Z axis</td>
</tr>
<tr>
<td>XTopAxis</td>
<td>Secondary X axis drawn at the top of the chart.</td>
</tr>
</tbody>
</table>
YRightAxis
Secondary Y axis drawn at the right of the chart.

SweepAxis
An axis used to sweep over an additional parameter.

**ChartColoringMethods**

Enumeration of the Coloring methods used for Tradeoff and Samples charts.

**Possible Values**

- ColoringPerFront
  A different color for each Pareto fronts (several samples have the same color).

- ColoringPerSample
  A different color for each sample.

**ChartStyle**

Allowed styles of a multi-axis chart.

**Possible Values**

- PCP
  Parallel Coordinate Plot

- Spider
  Spider Plot

**ChartType**

Enumeration for the chart type.

**Possible Values**

- ChartUnknown
  Unknown

- ChartResponse
  Response

- ChartTradeoff
  Tradeoff

- ChartSamples
  Samples

- ChartDistributions
  Distributions

- ChartCorrelation
  Correlation

- ChartSpiderResponses
  Spider

- ChartLocalSensitivity
  Local Sensitivity

- ChartSensitivities
  Sensitivities

- ChartStatistics
  Statistics

- ChartCorrelationScatter
  Correlation Scatter

- ChartDesignPointsParallel
  Parameters Parallel

- ChartDesignPointsCurves
  Design Points vs. Parameters

- ChartDetermination
  Determination Matrix

- ChartPredictedvsObserved
  Predicted vs. Observed

- ChartDeterminationHistogram
  Determination Histogram

- ChartConvergence
  Convergence

- ChartLocalSensitivityCurves
  Local Sensitivity Curves

- ChartCustom
  Custom Chart

- ChartCandidates
  Candidates

- ChartHistory
  History

- ChartConvergenceCriteria
  History

- ChartParameterRelationship
  History
**ClearanceType**

inpeller clearance type

**Possible Values**

- **Ratio**
  - tip clearance specified as a ratio
- **User**
  - tip clearance specified directly

**ClearanceType**

Enumeration of the tip clearance specification options.

**Possible Values**

- **None**
- **RelativeLayer**
- **AbsoluteLayer**

**Color**

This type represents a 32-bit Red/Green/Blue/Alpha color.

When working with Workbench Scripting, a color is represented as a four entry string in the form "R G B A", where each entry is in the range 0-255. Alpha (A) is optional and will be set to 255 (opaque) if not supplied. For example:

```python
renderStyle1.LineColor = "255 0 0"  # Sets the line to opaque Red.
renderStyle1.LineColor = "0 0 255 128" # Sets the line to translucent Blue.
```

**ComparePartsOnUpdateMethod**

Compare parts on update options

**Possible Values**

- **ComparePartsMethod_None**
  - Do NOT compare parts on update -- default value
- **ComparePartsMethod_Associatively**
  - Compare parts using associative mechanism, if geometry interface is non-associative expect failures in compare
- **ComparePartsMethod_NonAssociatively**
  - Compare parts using entity comparisons based on index of second model to original attach

**ComparePartsTolerance**

Compare parts on update tolerance options

**Possible Values**

- **ComparePartsTolerance_Loose**
  - A greater loosening of the default tolerance
Looser tolerance than default to allow some wiggle room for slight deviations
Default, existing behavior

**ConstraintHandlingType**

Enumeration of the Constraint Handling types.

**Possible Values**

- AsGoals: Relaxed constraint
- AsHardConstraints: Strict constraint

**ConstraintType**

Enumeration of the possible optimization constraint types.

**Possible Values**

- CT_NoPreference: No constraint defined.
- CT_NearTarget: Equals target.
- CT_LessThanTarget: Less than target.
- CT_GreaterThanTarget: Greater than target.
- CT_InsideBounds: Inside bounds.

**CoordinateSystemType**

Enumeration to specify the coordinate system type for imported data.

**Possible Values**

- Cartesian
- Cylindrical

**CorrelationAutoStopType**

Enumeration of the correlation Auto Stop types.

**Possible Values**

- ExecuteAllSimulations: Execute all Simulations.
- EnableAutoStop: Enable Auto Stop: execute Simulations iteratively until the process converges or the maximum number of simulation specified is reached.

**CoupledAnalysisType**

The valid coupling types
**Possible Values**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undefined</td>
<td>This is an internal option used to cache the user selection of analysis type. It cannot be specified by the user.</td>
</tr>
<tr>
<td>General</td>
<td>Define the coupling by time steps</td>
</tr>
<tr>
<td>Transient</td>
<td>Define the coupling by time intervals</td>
</tr>
<tr>
<td>Harmonic</td>
<td>Define the coupling by frequency intervals. The harmonic coupled analysis is not currently supported.</td>
</tr>
</tbody>
</table>

**DataContainerReference**

A reference to a data container, similar to a DataReference referring to an entity.

**DataOrder**

An enum of Row or Column which indicates if the Data is used in a row-major or column-major order. The default is Row.

**Possible Values**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
<td>Column-major</td>
</tr>
<tr>
<td>Row</td>
<td>Row-major</td>
</tr>
</tbody>
</table>

**DataReference**

A Data Reference holds and manages a reference to a data entity in the Workbench data model.

**DataReferenceSet**

This type contains an ordered set of DataReferences. No modifications can be made to the contents.

**DataTypeEnumeration**

This enumeration represents all of the tensor types supported by the MPC variable.

**Possible Values**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AType</td>
<td>An unknown type</td>
</tr>
<tr>
<td>Scalar</td>
<td>A scalar</td>
</tr>
<tr>
<td>VectorXY</td>
<td>A vector in x and y components</td>
</tr>
<tr>
<td>VectorYZ</td>
<td>A vector in y and z components</td>
</tr>
<tr>
<td>VectorXZ</td>
<td>A vector in x and z components</td>
</tr>
<tr>
<td>VectorXYZ</td>
<td>A vector in x, y, and z components</td>
</tr>
<tr>
<td>VectorRA</td>
<td>A vector in r and a components</td>
</tr>
<tr>
<td>VectorAZ</td>
<td>A vector in a and z components</td>
</tr>
<tr>
<td>VectorRZ</td>
<td>A vector in r and z components</td>
</tr>
<tr>
<td>VectorRAZ</td>
<td>A vector in r, a, and z components</td>
</tr>
<tr>
<td>VectorRI</td>
<td>A vector in r and i components</td>
</tr>
<tr>
<td>VectorIA</td>
<td>A vector in i and a components</td>
</tr>
<tr>
<td>VectorRIA</td>
<td>A vector in r, i, and a components</td>
</tr>
</tbody>
</table>
Tensor2XYZ
Tensor4XYZ
SymTensor2XYZ
SymTensor4XYZ
AntisymTensor2XYZ

A 2D tensor
A 4D Tensor
A 2D symmetric tensor
A 4D symmetric tensor
An anti-symmetric tensor

**DateTime**

This type represents a date and time. The default format for printing a DateTime object is "DD/MM/YYYY HH:MM:SS AMPM". Additional properties which can be examined on a DateTime object to extract extra detail include Year, Month, Day, Hour, Minute, Second and Date.

**DebugLevel**

Debug levels for the log output. Increasing levels generates more detailed output in the log.

**Possible Values**

- None: No debug output
- Level1: Level 1
- Level2: Level 2
- Level3: Level 3
- Level4: Level 4
- Level5: All Levels
- Default: Use Default Output Level

**DelimiterType**

Enumeration to specify the type of delimiter in imported data.

**Possible Values**

- Comma
- Semicolon
- Space
- Tab
- UserDefined

**DerivativeApproximationType**

Enumeration for the derivative approximation type.

**Possible Values**

- DA_CentralDifference: Central Difference
- DA_ForwardDifference: Forward Difference

**DeterminationCoefficientChartModes**

Enumeration of the Determination Coefficient chart modes.
**Possible Values**

- Linear
- Quadratic

Display linear determination coefficients.

Display quadratic determination coefficients.

**Dictionary<Key, Value>**

This type represents a data dictionary, where a Key is used to access an associated Value. When used in scripting, a dictionary is created or printed using the form

```python
myDict = {key1:value1, key2:value2, ...}
```

Python functionality can be used to examine dictionary keys, test for key existence and perform other useful operations on dictionaries.

The following example shows the use of dictionaries in a Workbench script:

```python
>>> template1 = GetTemplate(
    TemplateName="Fluid Flow",
    Solver="CFX")
>>> system1 = template1.CreateSystem()
>>> solution1 = system1.GetContainer(ComponentName="Solution")
>>> cfxSolutionProperties1 = solution1.GetCFXSolutionProperties()
>>> currentProps = cfxSolutionProperties1.GetEntityProperties()
>>> for key in currentProps.Keys:
    print "%s = %s" % (key, currentProps[key])

SolverCommandMode = Foreground
DisplayText = Solution Source
InitializationOption = CurrentSolutionData
LoadMResOptions = LastConfigOnly
ResultsFile = None
>>> myProps="{"SolverCommandMode":"Background","InitializationOption":"InitialConditions"}
>>> cfxSolutionProperties1.SetEntityProperties(Properties=myProps)
```

The above is equivalent to:

```python
# cfxSolutionProperties1.InitializationOption ="InitialConditions"
# cfxSolutionProperties1.SolverCommandMode="Background"
```

**DiffuserType**

diffuser type

**Possible Values**

- Vaned
- Vaneless

vaned diffuser
vaneless diffuser

**DimensionsType**

Enumeration to specify the dimensionality of imported data.

**Possible Values**

- Dimension2D
- Dimension3D
**DistType**

Enumeration of the supported distribution types for an input parameter.

**Possible Values**

- Uniform
- Triangular
- Gauss
- TruncGauss
- LogNorm
- Exponential
- Beta
- Weibull

**DotStyles**

Styles of dot symbols. Default is none

**Possible Values**

- None
- Ellipse
- Rect
- Diamond
- Hexagon
- Triangle
- DTriangle
- UTriangle
- LTriangle
- RTriangle
- Cross
- XCross
- Star
- Default

**double**

This type represents a double-precision floating point number.

**DurationType**

The methods used to specify duration

**Possible Values**

- NumberOfSteps The coupling will end at a given number of steps
- EndTime The coupling will end at a given time
EDesignPointScope

This describes specific groups of design points and is used by some commands that can work on a variable number of design points

- EPS_CURRENT - current design point in parameter manager
- EPS_PARAMETER_MANAGER - the parameter manager design point table
- EPS_DOE - parameter table built from all the DOE components that enable parameters associated with the system in question
- EPS_ALL - the combination of design points obtained by using the rules used for EPS_ALL and EPS_PARAMETER_MANAGER

**Possible Values**

- EPS_CURRENT
- EPS_PARAMETER_MANAGER
- EPS_DOE
- EPS_ALL

EffType

Impeller efficiency type

**Possible Values**

- Automatic  Efficiencies calculated from correlations
- Hydraulic  Hydraulic efficiency calculated. Volumetric, mechanical and overall pump efficiencies user defined.
- Volumetric  Volumetric efficiency calculated. Hydraulic, mechanical and overall pump efficiencies user defined.
- Mechanical  Mechanical efficiency calculated. Hydraulic, volumetric and overall pump efficiencies user defined.
- Pump  Overall pump efficiency calculated. Hydraulic, volumetric and mechanical efficiencies user defined.

EngineeringDataType

Supported types of engineering data.

**Possible Values**

- Unknown
- Material
- Load
- BeamSection
- Mixture

EtaCorrelType

Efficiency correlation type
Possible Values

Suhrmann
Baines

Suhrmann's correlation
Baines' correlation

EtaCorrelType

Efficiency correlation type

Possible Values

CaseyRobinson
CaseyMarty
Rodgers

Casey-Robinson correlation
Casey-Marty correlation
Rodgers correlation

EtaImpType

impeller isentropic efficiency type

Possible Values

LinkToStage
User

linked to stage efficiency
user specified efficiency

EtaType

Stage efficiency type

Possible Values

User
Correlation

User defined efficiency
Efficiency calculated from correlation

EtaType

Stage efficiency type

Possible Values

User
Correlation

User defined efficiency
Efficiency calculated from correlation

EtaUserType

User specified stage efficiency type

Possible Values

Isentropic
Polytropic

user-specified isentropic efficiency
user-specified polytropic efficiency
ExcelConnectionState

Enumeration for the Excel Connection states

**Possible Values**

- NotConnected: The connection with Excel is not established
- ConnectionAlive: The connection is alive
- ConnectionLost: The connection has been lost

ExecutionControlConflictOptions

Options for handling execution control conflicts on Edit

**Possible Values**

- Default
- UseSetupExecutionControl
- UseSetupExecutionControlAlways
- UseSolutionExecutionControl
- UseSolutionExecutionControlAlways

ExecutionControlSource

Enumeration for the execution control conflict resolution.

**Possible Values**

- IssueWarning
- UseExecutionControlFromSetup
- UseExecutionControlFromSolution

ExitAngleType

Exit angle type

**Possible Values**

- Absolute
- Relative

ExpressionType

Specifies the possible types of parameter expression.

**Possible Values**

- Undefined: An undefined(null) expression.
- Constant: An expression without dependency on other parameter.
- Derived: An expression with dependency on other parameters.
FileType

File Types recognized by FLUENT. Includes native FLUENT files as well as files that FLUENT can import or export. The Unknown File Type is used when FLUENT is unable to recognize the file type.

For more details on the file types recognized by FLUENT, please refer to the FLUENT User's Guide.

Possible Values

- Mesh
- Case
- Data
- Bc
- Pdf
- S2s
- Flamelet
- Dtrm
- Udf
- Udflib
- UdfSource
- Wave
- SurfaceMonitors
- VolumeMonitors
- ParticleInjections
- BoundaryProfile
- SolutionTranscript
- SchemeFile
- FluentResidualFile
- InterpolateDataFile
- registeredTypes
- AbaqusFilbin
- AbaqusInput
- AbaqusOdb
- Ansys
- AnsysResults
- CfxDefn
- CfxResults
- Cgns
- CgnsMeshData
- Ensight
- Fidap
- Fluent4
- FluentMesh
- Gambit
- Hypermesh
- IdeasUniv
- LstcInput
- LstcState
- MarcPost
- Mechanica
- MetisCase
- MetisZoneCase
Data Types

Nastran
NastranOutput
PamDaisy
Patran
PatranResults
Pda
Prebfc
Plot3dGrid
Plot3dResults
StlAscii
StlBinary
TecplotBinary
VkiGeneric
Unknown

**FillStyles**

Style of any filled region. The default is None.

**Possible Values**

- None
- Solid
- Dense
- Medium
- Sparse
- Horizontal
- Vertical
- Cross
- BDiagonal
- FDiagonal
- CrossDiagonal
- Gradient
- Default

**FittingType**

Enumeration for the Response Surface type.

**Possible Values**

- FITTINGTYPE_SRS  Standard Response Surface - Full second order Polynomials
- FITTINGTYPE_KRIGING  Kriging
- FITTINGTYPE_MARS  Non Parametric Regression
- FITTINGTYPE_NN  Neural Network
- FITTINGTYPE_SPARSEGRID  Sparse Grid

**FlowType**

Enumeration of the flow boundary condition options.
**Possible Values**

MassFlow  
PressureRatio  
PressureDifference

**FluidType**

Enumeration of the types of available fluids.

**Possible Values**

IdealGas  
RealGas  
Liquid

**FormatType**

Enumeration to specify the format type for imported data.

**Possible Values**

UserDefined  
Delimited  
Cdb  
Axdt

**GasModelType**

gas model type

**Possible Values**

Ideal  
Real  
Ideal gas model  
real gas model

**GasPropType**

Gas properties type

**Possible Values**

Air  
AFR  
Fixed  
Gas props - Air  
Gas props - Air/fuel ratio  
Gas props - Fixed

**GeometryAnalysisType**

3D/2D Geometry import option
**Possible Values**

- Type3D: Import all 3D objects
- Type2D: Import only 2D objects (The model must be in the x-y plane.)

**GeometryAttachType**

Type of geometry that is being attached.

**Possible Values**

- ThreeDimensional
- TwoDimensional
- Unknown

**GeometryStyleType**

Geometry export style type

**Possible Values**

- Interactive: Create interactive geometry
- Parametric: Create parametric geometry

**GoalType**

Enumeration of the possible optimization objective types.

**Possible Values**

- GT_NoPreference: No objective defined.
- GT_MaximumPossible: Maximize.
- GT_MinimumPossible: Minimize.
- GT_SeekTarget: Seek target.

**GPUAccelerator**

Enumeration for the graphics acceleration library to be used by the Mechanical APDL editor.

**Possible Values**

- None
- Amd
- Intel
- NVIDIA

**HubLEBetaType**

Hub and Mean LE blade angle option
**Possible Values**

- **Hub/Mean LE blade angle calculated using cotangent (rel to Shroud LE beta)**
  - **Cot**
- **Hub/Mean LE blade angle calculated using cosine (rel to Shroud LE beta)**
  - **Cos**
- **User defined hub and mean LE blade angles**
  - **User**

**ICCombustionSimulationType**

Type of IC Engine Combustion simulation type.

**Possible Values**

- **0 for sector**
  - **ICSector**
- **1 for full engine full cycle**
  - **ICFullEngineFullCycle**
- **2 for full engine IVC to EVO**
  - **ICFullEngineClosedValves**

**ICIVCandEVOoption**

IVC and EVO options

**Possible Values**

- **0 lift curv profile**
  - **ICLiftCurv**
- **1 for IVC EVO option**
  - **ICIVCandEVO**

**ICSimulationType**

Type of IC Engine Simulation.

**Possible Values**

- **0 for Cold Flow Simulation**
  - **ICSimulationColdFlow**
- **1 for Port Flow Simulation**
  - **ICSimulationPortFLow**
- **2 for Combustion Flow Simulation**
  - **ICSimulationCombustion**

**IDHandling**

The ID handling property for the current object.

**Possible Values**

- **No action must be carried out when renumbering the attributes of the current object.**
  - **None**
- **The IDs of the attributes of the current object must be automatically renumbered so that no conflicts will happen.**
  - **Automatic**
- **The ID Handling property is not set.**
  - **Unknown**
**ImpellerExportType**

Impeller export type

**Possible Values**

- Coupled
- Isolated

**ImpellerLengthType**

Impeller length ratio type

**Possible Values**

- Automatic
- User

**ImpellerType**

Impeller type

**Possible Values**

- Unshrouded
- Shrouded

**ImportanceLevel**

Enumeration of the importance levels which can be associated with an optimization objective or constraint.

**Possible Values**

- GL_MediumImportant
- GL_LowImportant
- GL_HighImportant

**IncidenceType**

Incidence type

**Possible Values**

- incidence
- choke

**InitializationMethods**

Initialisation Methods
**Possible Values**

ProgramControlled  
SolverControlled  
ProvideInitialSolution

**InitializationOption**

Enumeration for the Solution update initialization options.

**Possible Values**

CurrentSolutionData  
InitialConditions

**InitializationType**

The initialization settings

**Possible Values**

ProgramControlled  
StartTime  
Restart Step and Time

**InletAngleType**

Inlet angle type

**Possible Values**

Absolute  
Relative  
Calculated

**int**

This type represents an Integer number.

**long**

This type represents a long (64-bit) Integer number.

**JobRunMode**

The job running modes

**Possible Values**

Foreground  
Background  
Foreground mode.  
Background mode.
RemoteSolveManager Submitted to Remote Solve Manager to run the job.

KernelVariationType

Enumeration of the Kernel Variation types.

**Possible Values**

- VARIABLE
- CONSTANT

LinearCorrelationType

Enumeration of the Correlation types.

**Possible Values**

- Spearman
- Pearson

LineStyles

Styles of lines that can be displayed. The default is Solid.

**Possible Values**

- None
- Solid
- Dense
- Medium
- Sparse
- DashShort
- DashMedium
- DashLong
- DashDot
- DashDotDot
- DashDashDot
- Gradient
- Default

List<Type>

This type represents an unordered list of values.

MachineSpecification

List of computers to be used for a parallel FLUENT session. The list can be specified directly, or a hosts file containing the list can be specified.

For more details on how to specify the machines to be used for a parallel FLUENT session, please refer to the FLUENT User's Guide.
**Possible Values**

MachineList

FileName

### MachineType

Setup Entity enum definition for machine type

**Possible Values**

- Pump
- AxialCompressor
- CentrifugalCompressor
- Fan
- AxialTurbine
- RadialTurbine
- HydraulicTurbine
- Other
- Unknown

### MaterialNamesList

Database materials list Note that this is currently a fixed list which must correspond to the vistaFluids.xml database

**Possible Values**

- Air
- CarbonDioxide
- Hydrogen
- Methane
- Nitrogen
- Oxygen
- Parahydrogen
- Propylene
- R123
- R125
- R134a
- R141b
- R142b
- R245fa
- Water

### MaterialPropsType

material properties type

**Possible Values**

- Database: select material from database
- User: user specified material properties
MeshFileType

MeshFileType is used to identify the format of the mesh data file. This is usually the same as the application that generated the data file.

**Possible Values**

- CFX
- ICEM_CFD
- FLUENT
- POLYFLOW
- Unknown

MeshRestartMode

Indicates the type of mesh import we want: - no initialization from upstream system - select a mesh in a list of mesh files coming from an upstream polyflow system

**Possible Values**

- NoUpstreamMeshFile
- SingleMeshFile

MessageType

The valid message types.

**Possible Values**

- Information
- Warning
- Error
- Fatal
- Problems
- Standard
- Debug
- Progress
- News

MixedImportPref

Mixed import preference option for mixed dimension parts

**Possible Values**

- MixedImport_None
- MixedImport_Solids
- MixedImport_Surfaces
<table>
<thead>
<tr>
<th>ModelType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MixedImport_Lines</td>
<td>If mixed dimension part, import Lines only</td>
</tr>
<tr>
<td>MixedImport_SolidsAndSurfaces</td>
<td>If mixed dimension part, import Solids and Surfaces only</td>
</tr>
<tr>
<td>MixedImport_SurfacesAndLines</td>
<td>If mixed dimension part, import Surfaces and Lines only</td>
</tr>
</tbody>
</table>

**Possible Values**

<table>
<thead>
<tr>
<th>ModelType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abaqus</td>
<td>The model is formatted according to the ABAQUS standard.</td>
</tr>
<tr>
<td>CFX</td>
<td>The model is formatted according to the CFX standard.</td>
</tr>
<tr>
<td>CMDB</td>
<td>The model is stored in a Meshing database.</td>
</tr>
<tr>
<td>Fluent</td>
<td>The model is formatted according to the Fluent standard.</td>
</tr>
<tr>
<td>Icem</td>
<td>The model is formatted according to the Icem standard.</td>
</tr>
<tr>
<td>MechAPDL1CDB</td>
<td>The model is formatted according to the CDB standard, used in Mechanical APDL.</td>
</tr>
<tr>
<td>Nastran</td>
<td>The model is formatted according to the Nastran standard.</td>
</tr>
<tr>
<td>SimulationSetup</td>
<td>The model is stored in a Simulation database.</td>
</tr>
<tr>
<td>STL</td>
<td>The model is formatted according to the STL standard.</td>
</tr>
<tr>
<td>ACMO</td>
<td>The model is stored in an ACMO database.</td>
</tr>
<tr>
<td>MechAPDLRST</td>
<td>The model is formatted according to the RST standard, used in Mechanical APDL.</td>
</tr>
<tr>
<td>AbaqusResults</td>
<td>The model is formatted according to the results standard used in ABAQUS.</td>
</tr>
<tr>
<td>NastranResults</td>
<td>The model is formatted according to the results standard used in Nastran.</td>
</tr>
<tr>
<td>SamcefResults</td>
<td>The model is formatted according to the results standard used in Samcef.</td>
</tr>
<tr>
<td>FEModeler</td>
<td>Internally used to specify a FE Modeler to FE Modeler connection.</td>
</tr>
<tr>
<td>Unknown</td>
<td>The model type property is not set.</td>
</tr>
</tbody>
</table>

**MonitorChartType**

MonitorChartType enum: Residual or UserDefined

**Possible Values**

<table>
<thead>
<tr>
<th>MonitorChartType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Residual</td>
<td></td>
</tr>
<tr>
<td>UserDefined</td>
<td></td>
</tr>
</tbody>
</table>
**MPIType**

Enumeration for the MPI library to be used by the Mechanical APDL solver.

**Possible Values**

- Undefined
- PCMPI
- MSMPI
- IntelMPI

**MResOptions**

Enumeration for the load options for Multi-configuration Results.

**Possible Values**

- AllConfigsSingleCase
- AllConfigsSeparateCases
- LastConfigOnly

**NumSampType**

Enumeration to specify for the samples type for OSFD algorithm.

**Possible Values**

- SFD_CCD: CCD samples
- SFD_LINEAR: Linear model samples
- SFDPUREQUAD: Pure quadratic model samples
- SFD_CROSSQUAD: Full quadratic model samples
- SFD_USER: User-defined samples

**NuUserType**

Kinematic viscosity calculation type (obsolete)

**Possible Values**

- Sutherland: calculate viscosity from Sutherland's Law (using coeffs for air)
- User: user specified kinematic viscosity

**Object**

This type can represent any generic object. It is used when any type is a valid value.

**OpeningPositionMethod**

Enumeration of the inlet/outlet placement options.
**Possible Values**

- Manual
- AdjacentBlade

**OptimalSpaceFillingType**

Enumeration of the available design types for the Optimal Space Filling algorithm.

**Possible Values**

- SFDTYPE_MDIST: Max-Min Distance
- SFDTYPE_CL2: Centered L2
- SFDTYPE_MAXENT: Maximum entropy

**OrientationStyle**

Allowed orientation of a legend. Default is Vertical

**Possible Values**

- Vertical
- Horizontal
- Default

**Output<Type>**

The Output type is used in select instances where a method returns additional information in a method argument as well as the method return. These output arguments are typically optional, and an output variable must be declared before it is used. Once assignment has been made to an output variable, the return value can be evaluated by using the Get() method on the variable.

The following example shows the declaration and use of an output argument.

```python
>>> template1 = GetTemplate(TemplateName="EngData")
>>> system1 = template1.CreateSystem()
>>> engineeringData1 = system1.GetContainer(ComponentName="Engineering Data")
>>> matl1 = engineeringData1.GetMaterial(Name="Structural Steel")
>>> matlProp1 = matl1.GetProperty(Name="Density")
>>> matlProp1.SetData(
        Variables="Density",
        Values="-10 [kg m^-3"]")
>>> from Ansys.Core.Commands import Output
>>> outMsg = Output[str]()
>>> if not matl1.IsValid(Message=outMsg):
    print "Material is not valid for the following reason:"
>>>    print outMsg.Get()
Material is not valid for the following reason:
The value(s) for Density must be greater than zero.
```

**OutputFrequencyType**

The entity stores the options to specify frequency of writing result files
**Possible Values**

<table>
<thead>
<tr>
<th>None</th>
<th>No intermediate result files</th>
</tr>
</thead>
<tbody>
<tr>
<td>EveryStep</td>
<td>Every Coupling Step</td>
</tr>
<tr>
<td>StepInterval</td>
<td>At defined interval</td>
</tr>
</tbody>
</table>

**OutputSource**

Source of the output values, indicating the method used to obtain them.

**Possible Values**

<table>
<thead>
<tr>
<th>UserEdited</th>
<th>The output values are edited by the user.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation</td>
<td>The output values are obtained by a real sim-</td>
</tr>
<tr>
<td></td>
<td>ulation.</td>
</tr>
<tr>
<td>ResponseSurface</td>
<td>The output values are obtained by evaluating</td>
</tr>
<tr>
<td></td>
<td>a response surface approximation.</td>
</tr>
</tbody>
</table>

**ParameterizedEntityPropertiesCollection**

A ReadOnlyDictionary for parameterized properties. The keys are the data references to the entities that hold the properties, the values are the list (one or more) of the parameterized properties.

**ParameterNature**

Enumeration of the possible nature of a parameter.

**Possible Values**

<table>
<thead>
<tr>
<th>NatureContinuous</th>
<th>Continuous</th>
</tr>
</thead>
<tbody>
<tr>
<td>NatureUsability</td>
<td>Obsolete. Instead of defining a usability para-</td>
</tr>
<tr>
<td></td>
<td>meter, define a continuous parameter with</td>
</tr>
<tr>
<td></td>
<td>the UseManufacturableValues set to True.</td>
</tr>
<tr>
<td>NatureDiscrete</td>
<td>Discrete</td>
</tr>
</tbody>
</table>

**ParameterRelationshipType**

Enumeration of the possible parameter relationship types.

**Possible Values**

<table>
<thead>
<tr>
<th>PRT_LessThanOrEqualTo</th>
<th>Less Than or Equal To</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRT_GreaterThanOrEqualTo</td>
<td>Greater Than or Equal To</td>
</tr>
</tbody>
</table>

**ParameterUsage**

Specifies the possible ways a parameter can be used or set within the data model.

**Possible Values**

<table>
<thead>
<tr>
<th>Input</th>
<th>A parameter whose value will be used by the</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>data model.</td>
</tr>
</tbody>
</table>
ExpressionOutput  
An output parameter whose value is based on an expression. The parameter cannot be associated directly with the data model.

DirectOutput  
A parameter whose value will be provided directly by the data model. The parameter expression has to be undefined(null).

**ParameterValue**
Type of a parameter value.

**Possible Values**

- `ActualValue`  
The actual value of the parameter.

- `VariationToReference`  
The variation of the parameter with respect to the current reference point, as a decimal number.

**PeriodicSurfType**
Enumeration of the periodic surface options.

**Possible Values**

- `OnePiece`
- `ThreePieces`

**PIFType**
power input factor type

**Possible Values**

- `correlation`  
- `user specified`

**PositionType**
Specifies the possible positions of a system when it is moved or created. These positions are relative to an existing system, which is specified in a separate argument.

**Possible Values**

- `Default`  
  Default position. It is a new "child" of position, by system if given, or a new "root" system otherwise.

- `Left`  
  Positioned left to a system.

- `Right`  
  Positioned right to a system.

- `Above`  
  Positioned above a system.

- `Below`  
  Positioned below a system.
Data Types

**PostReportNamesType**

CFD Post reports

**Possible Values**

None
AxialCompressorReport
AxialCompressorRotorReport
CentrifugalCompressorReport
CentrifugalCompressorBladeRowReport
CentrifugalCompressorRotorReport
TurbineReport
TurbineRotorReport
FanNoiseReport
FanReport
GenericReport
HydraulicTurbineReport
HydraulicTurbineRotorReport
PumpReport
PumpImpellerReport
StatorReport
TurbineStatorReport
Custom

**PreswirlType**

Preswirl type

**Possible Values**

constant  Constant inlet angle
free       Free vortex
forced     Solid body rotation
linear     Linear variation of Vw

**ProcessorUnit**

Various Processor Unit options available for Microsoft Scheduler

**Possible Values**

Core
Socket
Node

**Quantity**

This class represents a physical quantity that can be measured. It holds a double value and a string that specifies the value's unit of measurement. The Value and Unit can be accessed individually as properties on this type, and a Quantity can be converted to new units.
Mathematical operations can also be performed on Quantities, and these operations calculate and enforce dimensional consistency between units. Note the results of mathematical operations are always converted into the project unit system.

**RampingType**

Enum providing ramping options.

**Possible Values**

- None: No ramping.
- Linear: Linear profile ramping.

**ReadOnlyDictionary<Key, Value>**

This type represents a read-only data dictionary, where a Key is used to access an associated Value. When used in scripting, a dictionary is created or printed using the form

```python
myDict = {key1:value1, key2:value2, ...}
```

Python functionality can be used to examine dictionary keys, test for key existence and perform other useful operations on dictionaries; however, the contents may not be manipulated unless the dictionary or its contents are cloned into a regular dictionary.

**RealGas**

Enumeration of the available real gas materials

**Possible Values**

- Air
- CarbonDioxide
- Hydrogen
- Methane
- Nitrogen
- Oxygen
- Parahydrogen
- Propylene
- R123
- R125
- R134a
- R141b
- R142b
- R245fa
- Water
- Custom

**RepositoryFileInfo**

Class to specify a repository file
ResponseChartModes

Enumeration of the available Response chart modes.

Possible Values

- Curve2D: 2D response chart where an output parameter is plotted versus an input parameter.
- Surface3D: 3D response chart where an output parameter is plotted versus two input parameters.
- Slices2D: 2D response chart where an output parameter is plotted versus two input parameters, on the X axis and the other varying over several curves.

ResponseSurfaceRefinementType

Enumeration for the Refinement type.

Possible Values

- REFINEMENT_NONE: None
- REFINEMENT_AUTO: Automated
- REFINEMENT_MANUAL: Manual

RotationType

Enumeration of the machine rotational direction options.

Possible Values

- RightHanded
- LeftHanded

RoughnessType

surface roughness type

Possible Values

- Machined: machined surface finish
- Cast: cast surface finish

RshSpecification

Client used to connect to the nodes in a cluster of LINUX machines.

'Other' is used for a custom connect command.

Possible Values

- RSH
- SSH
Other

**SampleGenType**

Enumeration for the Sampling type.

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMPLE_GEN_LHS</td>
<td>LHS</td>
</tr>
<tr>
<td>SAMPLE_GEN_WLHS</td>
<td>WLHS</td>
</tr>
</tbody>
</table>

**SamplesChartModes**

Enumeration of the available Samples chart modes.

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Candidates</td>
<td>Draw the samples and highlight the optimization candidates.</td>
</tr>
<tr>
<td>ParetoFront</td>
<td>Draw the samples using colors that represent their Pareto front.</td>
</tr>
</tbody>
</table>

**Scale**

Enum to define the scale of the axis.

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>Linear scale</td>
</tr>
<tr>
<td>CommonLog</td>
<td>Common or log base 10 scale</td>
</tr>
<tr>
<td>NaturalLog</td>
<td>Natural log scale</td>
</tr>
</tbody>
</table>

**SchedulerSpecification**

Various Job Schedulers available on Unix/Linux.

**Possible Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSF</td>
<td></td>
</tr>
<tr>
<td>SGE</td>
<td></td>
</tr>
<tr>
<td>PBSPro</td>
<td></td>
</tr>
</tbody>
</table>
SensitivityChartModes

Sensitivity chart modes.

**Possible Values**

- BarChart
- PieChart

ShrLEBetaType

Shroud LE blade angle option

**Possible Values**

- Incidence: Shroud LE blade angle calculated from specified incidence
- User: User defined shroud LE blade angle

ShroudDiameterType

Shroud diameter type

**Possible Values**

- Diameter: user specified shroud inlet diameter
- Angle: user specified shroud vane inlet angle
- Optimum: optimised shroud inlet diameter (minimum Mrel)

SimulationType

Enumeration for the simulation types of a parameter.

**Possible Values**

- DesignVariable: Design variable
- UncertaintyVariable: Uncertainty variable

float

This type represents a single-precision floating point number.

SixSigmaTableTypes

Enumeration of the Probability Table types.

**Possible Values**

- ProbabilityTable
- InverseProbabilityTable
- Quantile-percentile
- Percentile-quantile
**SolverRestartMode**

Indicates the type of restart.

**Possible Values**

- **NoRestartFile**
  In this mode, no restart file is used to initialize time scheme and fields.
- **SingleResultFile**
  In this mode, a result file is used to initialize fields.
- **CombineResultRestartFiles**
  In this mode, a result file is used to initialize fields and a restart file is used to initialize time scheme (starting time + derivate).
- **SingleCsvFile**
  In this mode, a csv file is used to initialize fields.
- **CombineCsvRestartFiles**
  In this mode, a csv file is used to initialize fields and a restart file is used to initialize time scheme (starting time + derivates).
- **ManyResultFiles**
  In this mode, a list of polyflow result files will be used to define the flow field on which we evaluate a transient mixing task (computation of a set of trajectories).
- **ManyCsvFiles**
  In this mode, a list of polyflow csv files will be selected for a conversion into other kinds of results for a future mixing task for example.

**SpanwiseDistributionType**

Spanwise distribution type

**Possible Values**

- **General**
  Blade exported using general spanwise distribution
- **Radial**
  Blade exported using radial spanwise distribution

**StackingType**

Stacking type

**Possible Values**

- **Radial**
  Radial stacking
- **Tan**
  Beta calculated from tangent
- **Sin**
  Beta calculated from sin

**Status**

The current calculation status of the Excel file.
Data Types

**Possible Values**

- **UpToDate**
  - The output parameters are up to date.
- **OutOfDate**
  - The values of the input parameters are modified and the values of the output parameters are not recalculated yet.
- **ErrorsWhenCalculating**
  - Errors occurred during the calculation in Excel.

**STLSDTAlgorithm**

SDT Algorithm Options

**Possible Values**

- **STLSDTAlgorithm_Curvatures**
  - Curvature based SDT algorithm
- **STLSDTAlgorithm_Angles**
  - Angle based SDT algorithm

**string**

This type represents a String value.

**SystemPropertyDictionary**

A dictionary holding system properties. The keys in the dictionary will be of the standard names defined in Ansys.ProjectSchematic.SystemPropertyNames. For each property, the value in the dictionary is a list of all unique values of that property.

**TipDiamType**

Impeller tip diameter option

**Possible Values**

- **Automatic**
  - D2 calculated automatically (from stability factor)
- **HeadCoeff**
  - D2 calculated from head coefficient
- **User**
  - D2 user defined

**TopologyType**

All of the possible values for topology.

**Possible Values**

- **Point**
  - A topological point
- **Curve**
  - A topological curve
- **Surface**
  - A topological surface
- **Volume**
  - A topological volume

**TradeoffChartModes**

Enumeration of the available Tradeoff chart modes.
Possible Values

- Curve2D: 2D tradeoff chart where a parameter is plotted versus another parameter.
- Surface3D: 3D tradeoff chart where a parameter is plotted versus two other parameters.

TransferAtType

Enum providing options of when to Transfer Data.

Possible Values

- StartOfStep: Transfer data at start of step.
- StartOfIteration: Transfer data at start of iteration.

TransferDataFromNewComponentSpec

Specifies the information needed to transfer data from a component that is being created in a new system.

TransferDataToNewComponentSpec

Specifies the information needed to transfer data to a component that is being created in a new system.

TransformationType

Enumeration of the available transformation types applicable to an output parameter.

Possible Values

- TransTypeNone: No transformation
- TransTypeBox: Box-Cox
- TransTypeYeo: Yeo-Johnson

TypeOfInitialSampling

Enumeration for the Initial Sampling Type.

Possible Values

- E_Screening: Screening
- E_OSF: Optimal Space-Filling
- E_ConstrainedSampling: Constrained Sampling

uint

This type represents an unsigned integer number. Negative values are invalid.

UpdatableEntityState

Types of state an updatable entity can have.
Possible Values

Unknown: entity state can not be defined.
OutOfRange: entity is out of date.
UpToDate: entity is up to date.
Error: last update of the entity gave an error.
OutOfRangeWithError: entity is out of date and last update of the entity gave an error.
UpToDateWithError: entity is up of date and last update of the entity gave an error.
RefreshRequired: entity needs a refresh.

UpdateErrorBehavior

Specifies the types of behavior if an error is encountered when updating multiple design points.

Possible Values

Stop: Terminate the update immediately on the first error.
SkipDesignPoint: Don't do any more work on the failing design point, but continue with the next one if any.
Continue: Update as much of the project, and as many design points, as possible.

VariableConversionOption

Enumeration to specify the conversion of data

Possible Values

NoConversion
AverageSharedNodes
AverageNodesToElement
AverageNodesToFace
DistributeElementToNodesEqually
DistributeFaceToNodesEqually
AverageCornerToMidsideNodes

VariableExposure

Level of exposure for variables in the CDI

Possible Values

Standard: Default
Expert: Expert

VariableStyle

Styles that can be used to display the variable. Default is Line
**Possible Values**

None
Line
Spline
Step
Bar
Default

**ViscosityType**

*viscosity type*

**Possible Values**

- Sutherland: Viscosity calculated using Sutherland's Law (2 coefficient method)
- Dynamic: User-specified dynamic viscosity
- Kinematic: User specified kinematic viscosity

**VoluteType**

Volute style option

**Possible Values**

- Elliptic: Elliptic/circular cross sections
- Rectangular: Rectangular cross sections

**XAxisQuantity**

Enum for quantity used for X-axis.

**Possible Values**

- Iteration
- FlowTime
- TimeStep

**YN**

This is enum for yes/no option

**Possible Values**

- ICYes
- ICNo
Index

A
Ansoft data container, 37
Ansoft Feedback Iterator data container, 39
Ansoft Namespace, 697
ANSYS Workbench journaling
overview, 1
ANSYS Workbench scripting
examples, 16
   APDL example, 25
data-integrated application, 25
embedded script, 25
file management, 20
material properties, 23
project updates, 16
tabular data, 23
upating from Excel, 29
file path handling, 14
object-based, 13
overview, 1
path handling, 14
units, 15
using, 13
APDL script scripting examples, 25
AQWA Model data container, 43
AQWA Results data container, 45
AQWA Setup data container, 46
AQWA Solution data container, 48
argument
   scripting definition, 10
AUTODYN Analysis data container, 51
AUTODYN Setup data container, 51

C
CFD Results data container, 55
CFX Setup data container, 59
CFX Solution data container, 62
command window
   journaling, 3
   navigation, 4
component
   scripting definition, 8
console window
   journaling, 6

D
data container
   scripting definition, 8, 13
data container reference
   scripting definition, 9

Data Containers
   Ansoft, 37
   Ansoft Feedback Iterator, 39
   AQWA Model, 43
   AQWA Results, 45
   AQWA Setup, 46
   AQWA Solution, 48
   AUTODYN Analysis, 51
   AUTODYN Setup, 51
   CFD Results, 55
   CFX Setup, 59
   CFX Solution, 62
   DX Direct Optimization, 67
   DX Evaluation Container, 110
   DX GDO Design of Experiment, 179
   DX GDO Response Surface, 200
   DX Parameters Correlation, 232
   DX Six Sigma Analysis, 249
   Engineering Data, 265
   Engineering Data Curve Fit, 299
   Engineering Data Favorite Items, 303
   Engineering Data Favorite Library, 324
   External Connection, 351
   External Data, 353
   External Model Setup, 367
   Finite Element Modeler, 375
   FLUENT Setup, 385
   FLUENT Solution, 398
   Geometry, 415
   Graphics, 425
   ICE, 443
   ICE Setup, 445
   ICEM CFD, 449
   IcePak Setup, 451
   IcePak Solution, 452
   Mechanical APDL, 455
   Mechanical Enhanced Model, 461
   Mechanical Model, 462
   Mechanical Results, 471
   Mechanical Setup, 472
   Mechanical Solution, 475
   Mesh, 481
   Microsoft Office Excel Analysis, 489
   Parameters, 497
   Polyflow Setup, 507
   Polyflow Solution, 511
   Project, 515
   Project File Types, 526
   Project Files, 527
   Project Messages, 532
   System Coupling Setup, 535
   System Coupling Solution, 553

793
Index

Turbo Geometry, 563
Turbo Mesh, 568
Units, 693
Vista AFD Analysis, 572
Vista AFD Design, 589
Vista AFD Meanline, 605
Vista CCD, 614
Vista CCM, 641
Vista CPD, 648
Vista RTD, 666
Vista TF Setup, 684
Vista TF Solution, 692
data entity
  scripting definition, 9, 13
data reference
  scripting definition, 9
Data Types, 753
data-integrated applications
  scripting, 6
data-integrated applications scripting examples, 25
DX Direct Optimization data container, 67
DX Evaluation Container data container, 110
DX GDO Design of Experiment data container, 179
DX GDO Response Surface data container, 200
DX Parameters Correlation data container, 232
DX Six Sigma Analysis data container, 249
E
EKM Namespace, 700
embedded script scripting examples, 25
EngData Namespace, 705
Engineering Data Curve Fit data container, 299
Engineering Data data container, 265
Engineering Data Favorite Items data container, 303
Engineering Data Favorite Library data container, 324
Extensions Namespace, 706
External Connection data container, 351
External Data data container, 353
External Model Setup data container, 367
F
file management scripting examples, 20
file path handling
  ANSYS Workbench scripting, 14
  absolute/relative paths, 14
  slashes, 14
Finite Element Modeler data container, 375
FLUENT Setup data container, 385
FLUENT Solution data container, 398
G
Geometry data container, 415
Graphics data container, 425
Graphics Namespace, 709
I
ICE data container, 443
ICE Setup data container, 445
ICEM CFD data container, 449
IcePak Setup data container, 451
IcePak Solution data container, 452
IronPython, 5
J
journaling
  command window, 3
  navigation, 4
  shortcuts, 4
  console window, 6
  definition, 1, 7
  overview, 1
  playing a journal, 2
  preferences, 1
  recording, 2
  uses, 1
K
keyboard shortcuts
  command window, 4
Known Issues and Limitations, 33
M
material properties scripting examples, 23
Mechanical APDL data container, 455
Mechanical Enhanced Model data container, 461
Mechanical Model data container, 462
Mechanical Namespace, 712
Mechanical Results data container, 471
Mechanical Setup data container, 472
Mechanical Solution data container, 475
Mesh data container, 481
Meshing Namespace, 712
method
  scripting definition, 9
Microsoft Office Excel Analysis data container, 489
N
namespaced command
  scripting definition, 10
Namespaced Commands, 697
Namespaces
  Ansoft, 697
  EKM, 700
  EngData, 705
Extensions, 706
Graphics, 709
Mechanical, 712
Meshing, 712
Parameters, 713
Project, 720

O
object
  scripting definition, 9

P
Parameters data container, 497
Parameters Namespace, 713
path handling
  ANSYS Workbench scripting, 14
  absolute/relative paths, 14
  slashes, 14
Playing a journal, 2
Polyflow Setup data container, 507
Polyflow Solution data container, 511
preferences
  journaling, 1
project
  scripting definition, 8
Project data container, 515
Project File Types data container, 526
Project Files data container, 527
Project Messages data container, 532
Project Namespace, 720
Project updates scripting examples, 16
property
  scripting definition, 9
Python, 5

Q
query
  scripting definition, 10

R
recording a journal, 2

S
scripting
  definition, 5
  argument, 10
  component, 8
  data container, 8
  data container reference, 9
  data entity, 9
  method, 9
  namespaced command, 10
  object, 9
  project, 8
  property, 9
  query, 10
  system, 8
  templates, 10
  definitions, 7
    data container, 13
    data entity, 13
    data reference, 9
  overview, 1
SendCommands, 6
  using, 13
    with data-integrated applications, 6
SendCommands
  using, 6
setting journaling preferences, 1
setting preferences
  journaling, 1
shortcuts
  command window, 4
system
  scripting definition, 8
System Coupling Setup data container, 535
System Coupling Solution data container, 553

T
tabular data scripting examples, 23
templates
  scripting definition, 10
Turbo Geometry data container, 563
Turbo Mesh data container, 568

U
units
  ANSYS Workbench scripting, 15
Units data container, 693
updating from Excel scripting examples, 29
using ANSYS Workbench scripting, 13
using the command window
  journaling, 3
  navigation, 4

V
Vista AFD Analysis data container, 572
Vista AFD Design data container, 589
Vista AFD Meanline data container, 605
Vista CCD data container, 614
Vista CCM data container, 641
Vista CPD data container, 648
Vista RTD data container, 666
Vista TF Setup data container, 684
Index

Vista TF Solution data container, 692