

Scenario and Result Representations

Technical Article

Abstract

This article's main goal is to help partners understand the data model of the Scenario and Result Representation PLM objects and identify the APIs necessary to navigate them.

In order to better depict the relationships, the data model and navigation behavior is described through the following two sub-sections:

- [The Scenario Manager](#)
- [The Results Manager](#)

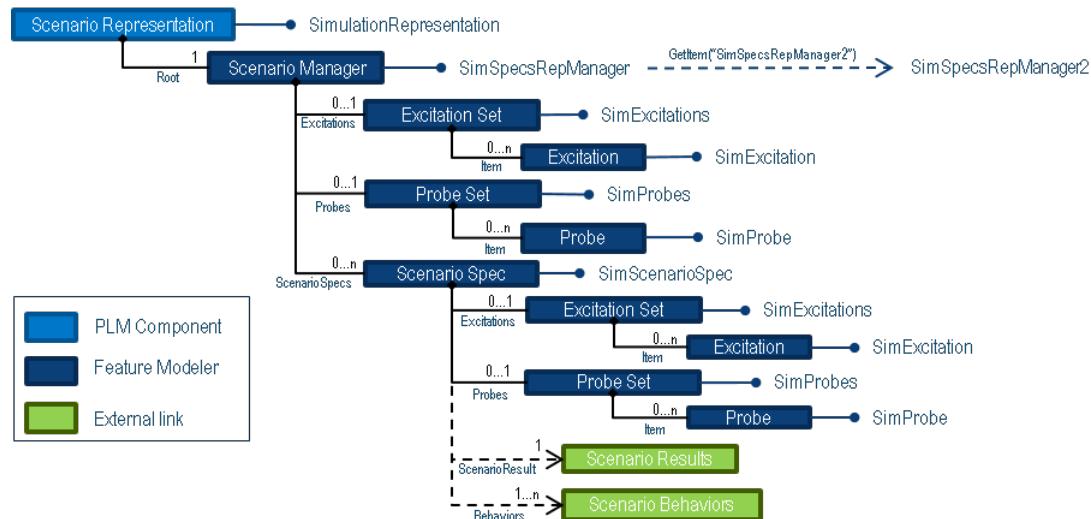
For further details on how to retrieve the scenario and result representation from a Simulation please refer to the use case "Navigation on a Simulation".[\[1\]](#)

The Scenario Representation Overview

The scenario representation contains a scenario manager. It is unique inside the representation and it contains all the pre-processing features necessary to describe the simulation scenario.

The data model and navigation APIs from the scenario representation are shown below:

Fig1: Scenario Representation to the Scenario Features

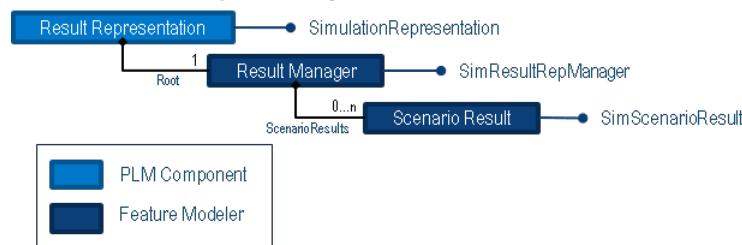


The Result Representation Overview

The results representation contains a result manager. It is unique inside the representation and it contains all the simulation results and the post-processing features necessary to describe the simulation results.

The data model and navigation APIs from the results representation is shown below:

Fig2: Results Representation to the Results Features



References

[1] [Navigation on a Simulation](#)

Assembly Evaluation Object Model Map

See Also [Legend](#)



Assembly Evaluation objects can be managed using the **FittingService** object. This object allows you to:

- Create or retrieve a **FitTrack** object.
- Manage **FitTrackTPoint** objects.

Use the **GetService** method of the **Service** object to return the **FittingService** object:

```
Dim oFittingService As FittingService
Set oFittingService = CATIA.ActiveEditor.GetService("FittingService")
```

Use the **Item** method of the **FittingService** object to return an existing **FitTrack** object:

```
Dim objFitTrack As FitTrack
Set objFitTrack = oFittingService.Item(objContext, 1)
```

Creating a Fitting Track in MSR

This use case primarily focuses on the methodology to create a Fitting Track in MSR Context.

Before you begin: Note that:

- You should first launch DELMIA and import the **CAAScdFitMSR.3dxml** files supplied in the folder **InstallRootFolder\CAADoc\Doc\English\CAAScdFit\samples** where **InstallRootFolder** is the folder where the CAA CD-ROM is installed.

Related Topics

Where to find the macro: [CAAScdFitUcTrackMsrSource.htm](#)

This use case can be divided in seven steps:

1. [Retrieves the current Editor](#)
2. [Retrieves the Fitting Service from the Editor](#)
3. [Creates a new Fit Track](#)
4. [Applies properties to Track](#)
5. [Retrieve the FitTrackTPoints object](#)
6. [Creates new TPoints](#)
7. [Refreshes the Track](#)

1. Retrieves the current Editor

```
...
Dim oEditor As Editor
Set oEditor = DELMIA.ActiveEditor
...
```

2. Retrieves the Fitting Service from the Editor

Then, retrieve the **FittingService** from Editor.

```
...
Dim oFittingService As FittingService
Set oFittingService = oEditor.GetService("FittingService")
...
```

3. Creates a new Fit Track

Then, Add a new Track.

Provide 'Nothing' as context in MSR context.

Provide 'DNBFitTrack' as type of Track.

```
...
Dim oContext As AnyObject
Set oContext = Nothing
Dim sTrackType As CATBSTR
sTrackType = "DNBFitTrack"
Dim oFitTrack As FitTrack
Set oFitTrack = oFittingService.Add(oContext, sTrackType)
...
```

4. Applies properties to Track

Apply the below properties to the newly creted Track:

Set Track Mode.

```
...
Dim eTrackMode As DNBTrackMode
eTrackMode = FitTIME
oFitTrack.Mode = eTrackMode
...
```

Set Track Interpolation Type.

```
...
Dim eTrackInter As DNBIInterpolater
eTrackInter = FitSPLINE
oFitTrack.Interpolator = eTrackInter
...
```

Set Track Total default Duration.

```
...
Dim dTotalTime As double
dTotalTime = 5.0
```

```

oFitTrack.TotalTime = dTotalTime
...

```

Set the Anchor Position to the Track. This is a set of twelve doubles.

Here we are setting the Anchor Position at the origin of the world.

```

...
Dim aAnchorPos(11) As double
aAnchorPos(0) = 1
aAnchorPos(1) = 0
aAnchorPos(2) = 0
aAnchorPos(3) = 0
aAnchorPos(4) = 1
aAnchorPos(5) = 0
aAnchorPos(6) = 0
aAnchorPos(7) = 0
aAnchorPos(8) = 1
aAnchorPos(9) = 0
aAnchorPos(10) = 0
aAnchorPos(11) = 0
oFitTrack.SetAnchorPosition aAnchorPos
...

```

5. Retrieve the FitTrackTPoints object

Retrieve FitTrackTPoints object which is a collection of TPoints.

```

...
Dim cFitTPoints As FitTrackTPoints
Set cFitTPoints = oFitTrack.TPoints
...

```

6. Creates new TPoints

Then, create TPoints with its positions. Here we will create three TPoints.

Note that the TPoints are relative to the Anchor Position.

First define the position of the First TPoint. Here, it is at the origin of the world.

```

...
ReDim aTPointPos(11) As double
aTPointPos(0) = 1
aTPointPos(1) = 0
aTPointPos(2) = 0
aTPointPos(3) = 0
aTPointPos(4) = 1
aTPointPos(5) = 0
aTPointPos(6) = 0
aTPointPos(7) = 0
aTPointPos(8) = 1
aTPointPos(9) = 0
aTPointPos(10) = 0
aTPointPos(11) = 0
...

```

Define the position of the Compass Offset from the TPoint. Here, there is no offset.

```

...
Dim TPointCompPos(11) As double
TPointCompPos(0) = 1
TPointCompPos(1) = 0
TPointCompPos(2) = 0
TPointCompPos(3) = 0
TPointCompPos(4) = 1
TPointCompPos(5) = 0
TPointCompPos(6) = 0
TPointCompPos(7) = 0
TPointCompPos(8) = 1
TPointCompPos(9) = 0
TPointCompPos(10) = 0
TPointCompPos(11) = 0
...

```

Insert the first TPoint with 'zero' duration.

```

...
cFitTPoints.InsertTPoint 1, aTPointPos, TPointCompPos, 0
...

```

Define the position of the second TPoint.

```

...
aTPointPos(9) = 100
aTPointPos(10) = 0
aTPointPos(11) = 0
...

```

Insert the second TPoint with duration 2.5.

```

...
cFitTPoints.InsertTPoint 2, aTPointPos, TPointCompPos, 2.5
...

```

Define the position of the third TPoint.

```

...
aTPointPos(9) = 100
aTPointPos(10) = 100
...

```

```

aTPointPos(11) = 0
...
Innsert the third TPoint with duration 2.5.

...
cFitTPoints.InsertTPoint 3, aTPointPos, TPointCompPos, 2.5
...

```

7. Refreshes the Track

Refresh/update the Track with all the above set properties and TPoints.

THIS IS A MANDATORY STEP. THE REFRESH MUST BE CALLED AT THE END OTHERWISE THE MODIFICATIONS WILL NOT BE REFLECTED ON THE TRACK.

```

...
oFitTrack.Refresh
...

```

Robotics Programming Overview

This article provides the basis for understanding how to customize the Robotics Offline Programming (OLP) translation.

OLP translation is the process of converting a DELMIA robot task into a native robot program for a specific robot programming language or visa-versa. This translation software is written using VB.NET VSTA macro libraries. You have access to the VB.NET source code for the language translation process which means that you can customize any aspect of the translation or even write a translator for which a DELMIA translator is not available.

For example, you may wish to modify how a robot program is created from a DELMIA task (download) in order to:

- Add comments to the top of a program.
- Add a special instruction before every spot operation.
- Modify how a specific instruction is translated. For example, you could generate the text for your own custom robot instruction for every robot motion instead of the standard motion syntax.
- Validate that the robot program conforms to your template and programming guidelines.

When importing a robot program into DELMIA (upload) you may wish to:

- Create DELMIA instructions for non-standard robot instructions.
- Modify attributes of DELMIA instructions when they are created.

Download: The process of generating a robot program text file from one or more DELMIA robot tasks.

Upload: The process of creating a DELMIA robot task from a robot program text file.

Before you Start

This documentation assumes that you are already familiar with the OLP Upload and Download commands. You can find information about those commands in the user documentation under "Content and Simulation Apps | Robotics | Robot Programming | Creating and Importing Robot Programs".

We also assume that you are familiar with VB, and specifically VB.NET and object oriented programming.

Getting Started

To get started with customizing an OLP translator, we recommend following these steps.

1. Read this document, the OLP Overview. It includes background information on the translation process and the technologies and objects used during translation.
2. Follow the use case for [Customizing a Translator](#). This use case provides an overview of the steps involved in customizing a translator.
3. Follow the use case for [Translating an Instruction](#). This provides the detail from [step 4 in Customizing a Translator](#). It walks you through the code which translates an instruction.
4. Follow the use case for [Integrating Your Customizations](#). This provides the detail from [step 5 in Customizing a Translator](#). It walks you through the steps for calling the code written in [Translating an Instruction](#).

Translation Architecture

The OLP Translators are VSTA macro libraries written in VB.NET. The macros are executed during the Download and Upload translation processes. In V6 all macro libraries, including the OLP translators, are stored in the ENOVIA database. This means you have access to the translation source code to view and modify as needed. Also because it is in the database, your modifications will be available to your entire enterprise.

The translation is a multi-step process.

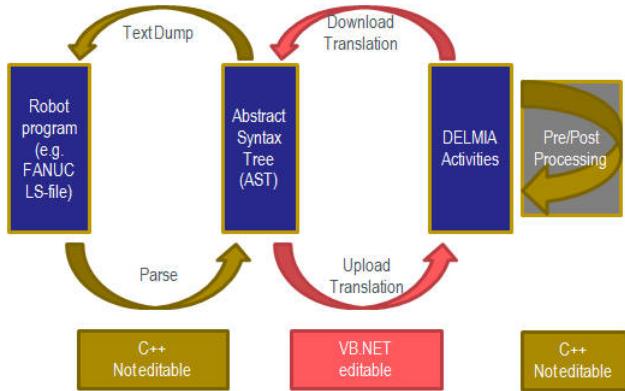
Download

1. Pre-process the DELMIA task(s) and robot controller to initialize the OLP automation objects.
2. Run the Download macro to generate the Abstract Syntax Tree (AST)
3. Dump the AST into the text file.

Upload

1. Pre-process the DEMIA robot controller to initialize the OLP automation objects.
2. Run the Upload macro. This macro has 2 jobs
 1. Call the [OlpParser](#) object to parse the file. It returns the Abstract Syntax Tree (AST).
 2. Convert the AST into DELMIA task (s)
3. Post-process the created tasks.

Fig.1: Translation Process



The translator macro library converts between 2 data structures, the Abstract Syntax Tree (AST) and the DELMIA objects. The AST is a parsed version of the robot program text file. Access to the DELMIA objects is provided through the [OLP Automation objects](#).

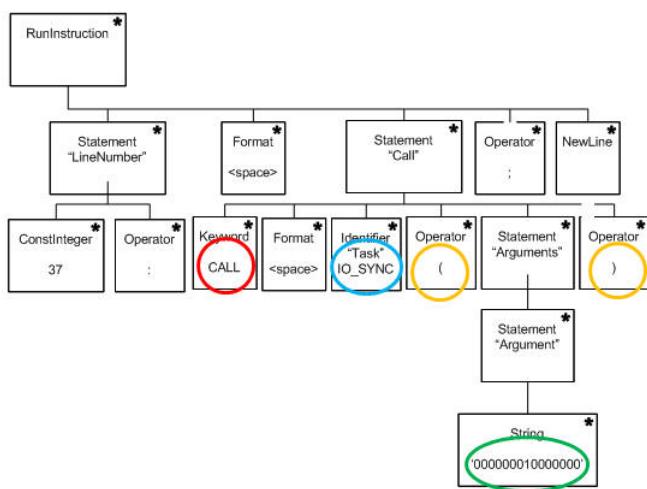
The parsers are written in C++. You do not need to modify them unless you are creating a translator to upload a language for which no DELMIA translator exists. If you need to create a new parser, you could write the parser in VB.NET to create the AST. The parsers have been written to accept any valid syntax for the robot language. In a procedure-oriented language like ABB RAPID, the AST structure for a call will be created for any unknown instructions. In other languages that do not have a well defined generic instruction grammar, like FANUC TP, the text of any instruction which cannot be parsed, like a macro call, will be stored in the AST as a generic statement.

Abstract Syntax Tree (AST) Data Structure

The AST is organized into a tree format. You can programmatically access each portion of an instruction based on its meaning. By traversing the tree, starting at the 1st leaf node and proceeding to the last node, the text of the robot program can be constructed. Below is an example of how a FANUC TP run instruction is represented in AST.

Fig.2: AST Example

37 : CALL IO_SYNC ('0000000100000000') ;



The AST is composed of 2 types of nodes. Branches, represented by [OlpAstBranch](#), are nodes with children. Leaves, represented by [OlpAstLeaf](#), contain the individual tokens from the robot program. Each node has 3 pieces of data associated with it.

- Type - This identifies the category of data stored in the node. For example Header, Instruction, Keyword, Comment, Operator, etc.
- ID - This identifies the purpose of the node. For example Speed, LineNumber, etc.
- Value OR Children
 - Value - Leaf nodes have a value stored on them which is the text from the robot program.
 - Children - Branch nodes have a list of sub-nodes.

The ID and Type can be used to easily locate a particular piece of data in an AST instruction.

Translator Macros

There are several macros in an OLP translator VB.NET macro library. These macros are called during different times of the translation process. Some macros are optional and are ignored if missing.

Macro Name	Description
MacroDownload	Called after the download pre-process step to translate the DELMIA objects into the AST.
MacroUpload	Called after the upload pre-process step. It parses the programs using OlpParser and then translates the AST into DELMIA objects.
MacroGetTranslatorInfo	(Optional) Called before a download or upload to get basic translator information. This includes the file extensions used for this robot programming language. This may also be used to identify features supported by a particular translator.
MacroSetConfigs	(Optional) Called during the upload post-process step. Some languages require inverse-kinematics calculations in order to compute the robot posture and turn numbers or turn signs. The post process step includes positioning tags in the DELMIA simulation which is required before

inverse-kinematics can be performed. After the tags have been correctly located, MacroSetConfigs is called, if it exists, to allow the translator to compute the robot posture and turn information.

OLP Automation Objects

During translation, access to the DELMIA task, instructions, and robot controller is provided through the OLP Automation objects. [OlpTranslatorHelper](#) is the main object which gives you access to all the objects required for translation. See the [Robotics Programming Object Model Map](#) and the [OLP Automation Object Reference](#) for more details.

Warning: The OLP automation objects can only be used by a macro which is called from the OLP Download or Upload commands. These objects do not exist outside the scope of an OLP translation.

Warning: Use of automation objects other than the OLP automation objects could cause problems during upload or download. In general, you should not use any automation object for which there is an equivalent OLP object. For example, you should not use any other objects which manipulate tasks, instructions, variables, robot profiles, or robot controllers. There are a few exceptions to these rules. The following APIs are approved for use in an OLP translator:

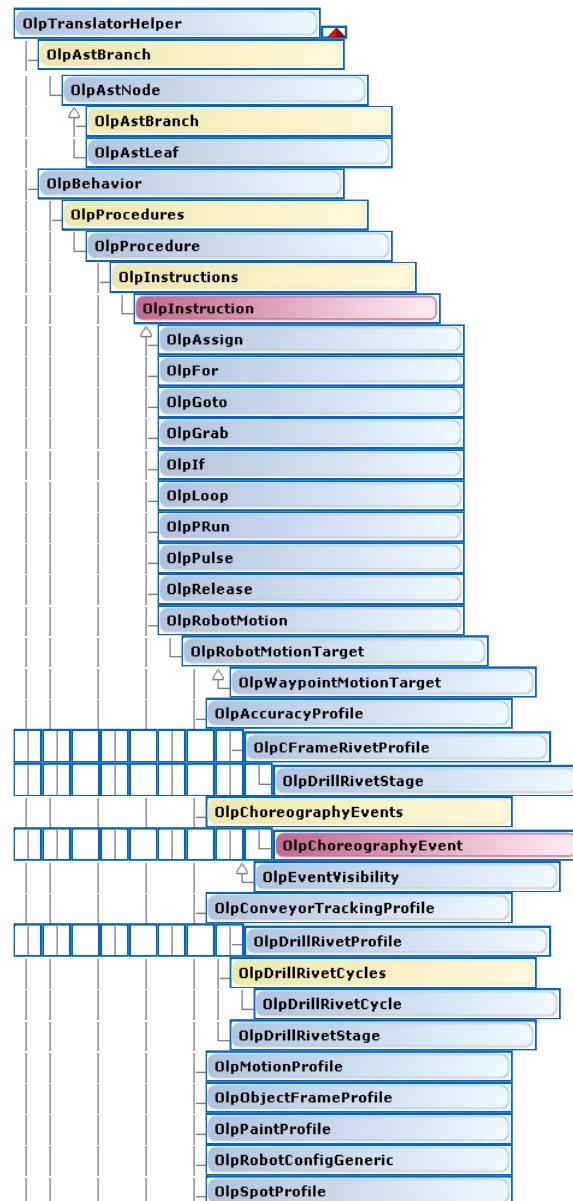
- RscApplicativeProfilesMgr
- RscApplicativeProfilesGroup
- RscApplicativeProfile
- RscMotionProfile
- RscAccuracyProfile
- ArcWeldProfile
- ArcWeldProfileSection
- ArcWeldCSPProfiles

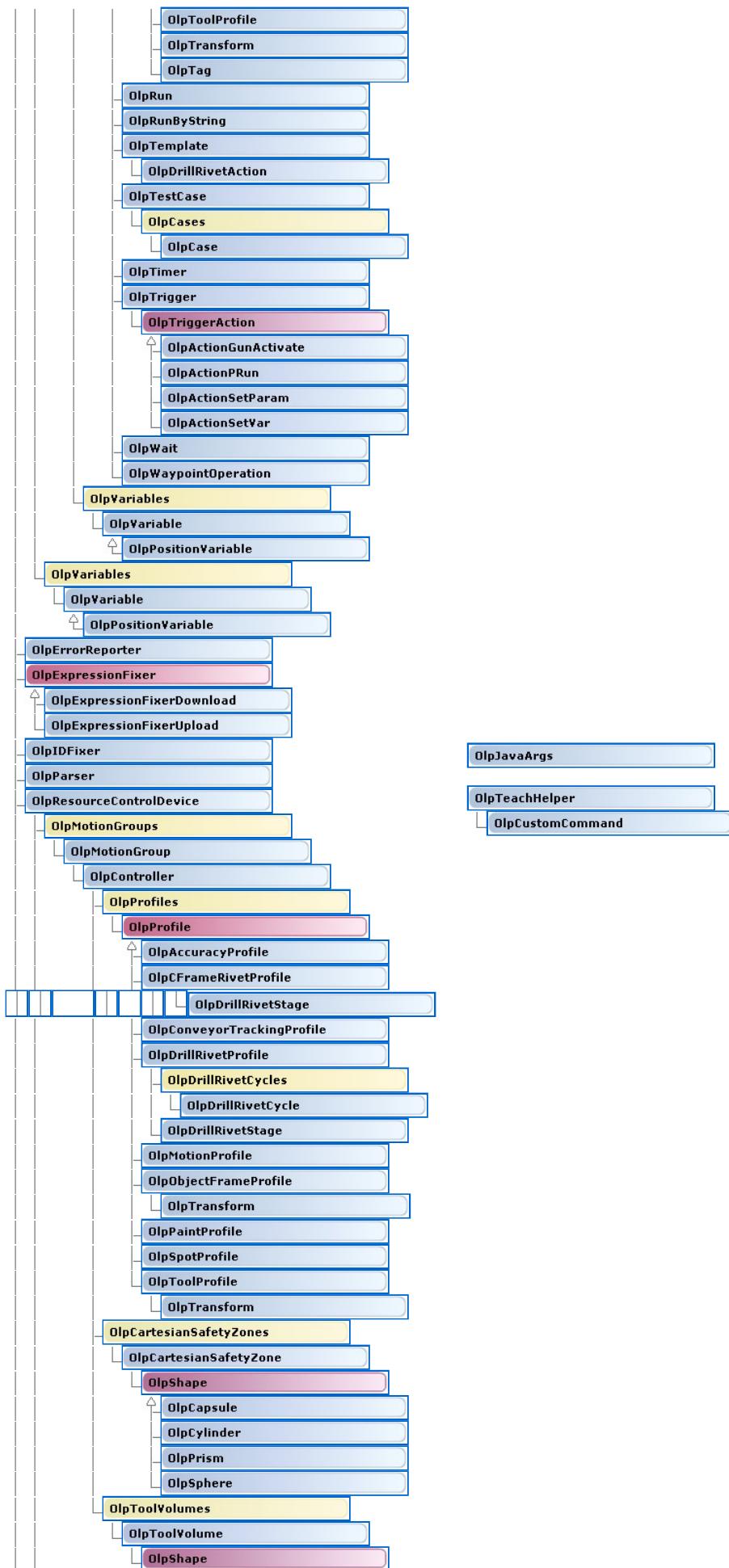
Additional Information

For details on translating the expressions used in DELMIA instructions, please see [OLP Expression Translation](#).

Robotics Programming Object Model Map

See Also [Legend](#)



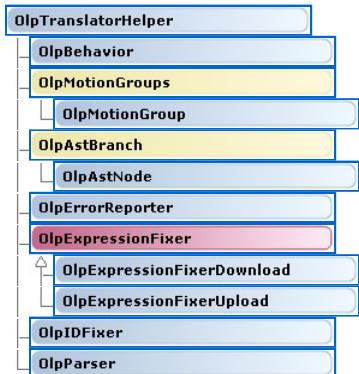




Note: there is no actual inheritance between **OlpAstBranch**, **OlpAstLeaf**, and **OlpAstNode**. This is because **OlpAstBranch** is a collection and **OlpAstLeaf** is an object. However, all objects which are an **OlpAstLeaf** are also a **OlpAstNode** and all objects which are an **OlpAstBranch** are also an **OlpAstNode**.

OlpTranslationHelper Object

See Also [Legend](#) Use Cases [Properties](#) [Methods](#)



The **OlpTranslationHelper** object is the root object for Robotics Programming (Olp) applications. It is a service object retrieved from the **Application** object thanks to the **GetSessionService** method, as follows:

```
Dim oOlpTranslationHelper As OlpTranslationHelper
Set oOlpTranslationHelper = CATIA.GetSessionService("OlpTranslationHelper")
```

Using the OlpTranslationHelper Object

Use the **Behavior** property to return the **OlpBehavior** object that itself aggregates the global variables and procedures as an **OlpVariables** and an **OlpProcedures** collection objects respectively.

```
Dim oOlpBehavior As OlpBehavior
Set oOlpBehavior = oOlpTranslationHelper.Behavior
```

Use the **MotionGroups** property to return the **OlpMotionGroups** collection object that contains the motion groups selected for download or upload.

```
Dim oOlpBehavior As OlpBehavior
Set oOlpBehavior = oOlpTranslationHelper.Behavior
```

Use the **AstRoot** property to return the **OlpAstBranch** collection object that represents the root node of the abstract syntax tree.

```
Dim oOlpRoot As OlpAstBranch
Set oOlpRoot = oOlpTranslationHelper.AstRoot
```

Use the **ErrorReporter** property to return the **OlpErrorReporter** object that report errors, warnings; and so on.

```
Dim oOlpErrorReporter As OlpErrorReporter
Set oOlpErrorReporter = oOlpTranslationHelper.ErrorReporter
```

Use the **CreateExprFixerDownload** method to create a new expression for download as a **OlpExpressionFixerDownload** object.

```
Dim oOlpExpressionFixerDownload As OlpExpressionFixerDownload
Set oOlpExpressionFixerDownload = oOlpTranslationHelper.CreateExprFixerDownload
```

Use the **CreateExprFixerUpload** method to create a new expression for upload as a **OlpExpressionFixerUpload** object.

```
Dim oOlpExpressionFixerUpload As OlpExpressionFixerUpload
Set oOlpExpressionFixerUpload = oOlpTranslationHelper.CreateExprFixerUpload
```

Use the **CreateIDFixer** method to create a new id fixer as an **OlpIDFixer** object.

```
Dim oOlpIDFixer As OlpIDFixer
Set oOlpIDFixer = oOlpTranslationHelper.CreateIDFixer
```

Use the **CreateParser** method to create a new robot program parser as an **OlpParser** object.

```
Dim oOlpParser As OlpParser
Set oOlpParser = oOlpTranslationHelper.CreateParser
```

OLP Expression Translation

This article provides details on translating expressions and the expression AST format used by the OLP Automation objects.

Many of the OLP Instructions have an expression as a property. An [Abstract Syntax Tree \(AST\)](#) is used to describe the expression. An AST is simply a parsed copy of the code so the AST format for one language is different from other languages. The OLP Expression AST format is used for DELMIA expressions. Please see the "DELMIA | Automation | Module & Block for Library | Reference Information | Predefined Types and Functions" in the user documentation for a description of the operators and function which are valid in a DELMIA expression.

The following instructions are examples of the ones that use expressions:

- [OlpAssign](#)
- [OlpFor](#)

- [OlpIf](#)
- [OlpLoop](#)
- [OlpRun](#)
- [OlpWait](#)

The translator needs to convert between the robot language expression and the DELMIA expression. In the translator both are represented in an AST format, so the translator needs to just make modifications to the AST structure to do the translation. There are 2 classes which help with this translation: [OlpExpressionFixerDownload](#) and [OlpExpressionFixerUpload](#). Please see the reference documentation for how to use those classes.

In general, the translator is responsible for the entire conversion from the Robot Language expression syntax to the DELMIA expression syntax. Using the expression fixer makes some of these transformations easier and validates that the expression conforms to the target language's limitations. There are a few cases where the OLP IDL interfaces directly manipulate and transform the expression such as for data type conversion and unit conversions.

Data Type Conversion: All the OLP instruction properties and methods where an expression can be set or retrieved, automatically add any data type conversion functions (e.g. NumberToBoolean, DoubleToInteger) that are necessary to prevent logic errors. Those data type conversion functions are removed automatically in the expression retrieved by the translator. For string concatenation, DELMIA uses the "++" operator. The "++" operator for string concatenation is always converted to "+" when the translator retrieves an expression.

Unit Conversions: Likewise, all the OLP instruction properties and methods automatically add or remove unit conversion functions if the robot language units differ from the DELMIA units. DELMIA always uses meters and radians for the units. The translator can specify the units used by the robot language by setting the properties PositionVariableRobotLanguageLengthUnits, PositionVariableRobotLanguageRotationUnits, and TrigFunctionRobotLanguageRotationUnits on [OlpTranslatorHelper](#).

Expression AST Format

The OlpExpressionFixer objects assume require that the AST conforms to a standard expression structure. The OLP Automation objects which use expressions also expect the expressions to be in this format when it is assigned. Expressions retrieved from the OLP Automation objects are also in this AST format.

The top level node of an expression of type *Expression*. An expression can contain any of the following nodes. Any of the following nodes can be used interchangeably in an expression. For example, anywhere that a *ConstInteger* is allowed you could have an *Identifier* or even a *BinaryExp*. Formatting nodes can be contained anywhere in the expression and are completely ignored when translating an expression but the formatting is preserved.

Leaf Nodes

ConstInteger	an integer number
ConstDouble	a floating point number
ConstBoolean	a Boolean
String	a string
Identifier	a variable name

Branch Nodes

BinaryExp	An expression involving 1 operator with 2 operands
UnaryExp	An expression involving 1 operator with just 1 operand
GroupExp	A grouping expression (parentheses).
Variable	A compound variable
Function	A function call
ConditionalExp	A conditional (if) expression.

Here is a sample AST for the simple expression A+B

```
<Expression>
  <BinaryExp>
    <Identifier id="Operand1">A</Identifier>
    <Operator>+</Operator>
    <Identifier id="Operand2">B</Identifier>
  </BinaryExp>
</Expression>
```

Leaf Expression Parts

These can be used in the place of any expression. For example A+B and A+1 are both valid. Technically A+true is a valid AST format but the resulting expression would be in error.

Integer

Any integer in a format allowed by DELMIA: 1, 0x1A, 0b1001, etc.

```
<ConstInteger>100</ConstInteger>
```

Double

Any double in a format allowed by DELMIA: 1.0, 3.14, 1e10, etc.

```
<ConstDouble>-2.3E5</ConstDouble>
```

Boolean

A Boolean value: "true" or "false".

```
<ConstBoolean>true</ConstBoolean>
```

String

A string, including the double quotes.

```
<String>"hello"</String>
```

Identifier

A variable name.

```
<Identifier>A</Identifier>
```

Branch Expression Parts

Each of these is an expression. An expression is composed of other expressions. Between any 2 nodes there can be a "Format" node.

Binary Expression

An expression involving 1 operator with 2 operands in the format "expression operator expression". Expressions are grouped in execution priority order. For example A+B+C would be grouped as ((A+B)+C). The expressions must have the "operand1" and "operand2" IDs set in order to easily identify the operands.

A+B+C is expressed as:

```
<BinaryExp>
  <BinaryExp id="Operand1">
    <Identifier id="Operand1">A</Identifier>
    <Operator>+</Operator>
    <Identifier id="Operand2">B</Identifier>
  </BinaryExp>
  <Operator>+</Operator>
  <Identifier id="Operand2">C</Identifier>
</BinaryExp>
```

Unary Expression

An expression involving 1 operator with just 1 operand in the format "operator expression". For example "-X". The expression must have the ID set to "Operand".

-X is expressed as:

```
<UnaryExp>
  <Operator>-</Operator>
  <Identifier id="Operand">X</Identifier>
</UnaryExp>
```

Group Expression

A grouping expression in the format "(Expression)". The expression must have the ID "Expression" set.

(X+1) is expressed as:

```
<GroupExp>
  <Operator>(</Operator>
  <BinaryExp id="Expression">
    <Identifier>X</Identifier>
    <Operator>+</Operator>
    <ConstInteger>1</ConstInteger>
  </BinaryExp>
  <Operator>)</Operator>
</GroupExp>
```

Variable Expression

A variable expression can be more complicated than a single identifier in the case of subscripted variables (e.g. x[1]) or data member access (e.g. P.x).

x[1] is expressed as follow. If FixSubscriptedVariables=True (the default) then this variable node (e.g. "x[1]") is treated as a unique variable including the subscripts in the Variables list of the expression fixer. If FixSubscriptedVariables=False then the array variable identifier (e.g. "x") is added to the Variables list.

```
<Variable>
  <Identifier>x</Identifier>
  <Operator>[</Operator>
  <Arguments>
    <ConstInteger id="Argument">1</ConstInteger>
  </Arguments>
  <Operator>]</Operator>
</Variable>
```

P.x is expressed as follows where each side of the "." operator must be tagged with the ID "Operand1" or "Operand2". Operand1 could be a subscripted variable in the structure as described above. Operand2 could be a identifier in the case of simple data member access, a function (e.g. inv()), or another variable node (e.g. j[2])

```
<Variable>
  <Identifier id="Operand1">varL1</Identifier>
  <Operator>.</Operator>
  <Identifier id="Operand2">x</Identifier>
</Variable>
```

Function

A function call in the format " function (expression , expression) ". There can be 0 or more expressions in the argument list. Each argument has the id set to "Argument" because it can be of any expression type. The function branch must contain an "Identifier" for the function name and an Arguments branch. The arguments branch can be empty. The parentheses around the arguments are specified as operator leaf nodes.

max(theta,x) is expressed as:

```
<Function>
  <Identifier>max</Identifier>
  <Operator>(</Operator>
  <Arguments>
    <Identifier id="Argument">theta</Identifier>
    <Operator>,</Operator>
    <Identifier id="Argument">x</Identifier>
  </Arguments>
  <Operator>)</Operator>
</Function>
```

Conditional Expression

A conditional expression in the format "if expression then expression else expression". The *else* is required. The *Condition*, *ThenExp*, and *ElseExp* IDs must be set.

if x=2 then 100 else 200 is expressed as:

```
<ConditionalExp>
  <Keyword>if</Keyword>
  <BinaryExp id="Condition">
    <Format> </Format>
    <Identifier>x</Identifier>
    <Operator>=</Operator>
    <ConstInteger>2</ConstInteger>
  </BinaryExp>
  <Format> </Format>
  <Keyword>then</Keyword>
  <Format> </Format>
  <ConstInteger id="ThenExp">100</ConstInteger>
  <Format> </Format>
  <Keyword>else</Keyword>
  <Format> </Format>
  <ConstInteger id="ElseExp">200</ConstInteger>
</ConditionalExp>
```

Complicated Example

Below is a more complex example of nested expressions. The expression below is $(X+\text{mod}(X,Y))/Y$.

```
<Expression>
  <BinaryExp>
    <GroupExp id="Operand1">
      <Operator></Operator>
      <BinaryExp id="Expression">
        <Identifier>x</Identifier>
        <Operator>+</Operator>
        <Function>
          <Identifier>mod</Identifier>
          <Operator>(</Operator>
          <Arguments>
            <Identifier>X</Identifier>
            <Operator>,</Operator>
            <Identifier>Y</Identifier>
          </Arguments>
          <Operator>)</Operator>
        </Function>
      </BinaryExp>
      <Operator></Operator>
    </GroupExp>
    <Operator>/</Operator>
    <Identifier>Y</Identifier>
  </BinaryExp>
</Expression>
```

Error Cases

An expression can be empty, which will result in an error during simulation or it can contain only formatting, which is also an error. If there is a problem parsing a DELMIA instruction on download the invalid expression text is stored as formatting when the expression is retrieved from the instruction. The following are examples of error cases which could be produced.

```
<Expression></Expression>

<Expression>
  <Format>Invalid Expression+</Format>
</Expression>
```

Customizing a Translator

This example illustrates how to modify a translator to convert a non-standard robot language instruction into a DELMIA instruction and back again.

Before you begin:

- You should have read [Robotics Programming Overview](#).
- You can import a working version of the customized translator from the file `MyFanucTranslatorR2016x.3dxml` supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScd0lpofflineProg\samples` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed. That folder also has the sample FANUC robot program `MAIN.txt`.

This example steps through how to customize the FANUC translator. To illustrate the example, we have invented a custom FANUC instruction "`IO_SYNC`".

```
CALL IO_SYNC('0000000100000000') ;
```

`IO_SYNC` is a procedure to do IO handshaking. It sets a group output signal and waits for a group input signal. The group output is set to the string argument. The string is interpreted as a binary number. After setting the output, it waits for the input value to be equal to the output value.

We are assuming that `IO_SYNC` is not a normal FANUC task. For example, it could be a KAREL program. This is a language that is sometimes used to write custom procedures for the FANUC controller. Each robot manufacturer offers a different way to customize their language. The basic philosophy for how to customize a DELMIA translator is the same for all languages.

The FANUC translator without modification creates `IO_SYNC` as an empty procedure. It creates one input argument to that procedure. Then it creates a run instruction to call that procedure and pass in the input value. The result is that nothing happens for this instruction during simulation because the `IO_SYNC` task has no instructions. The goal of this customization is to have IO handshaking executed during simulation.

Related Topics
[Robotics Programming Overview](#)
[Translating an Instruction](#)
[Integrating Your Customizations](#)

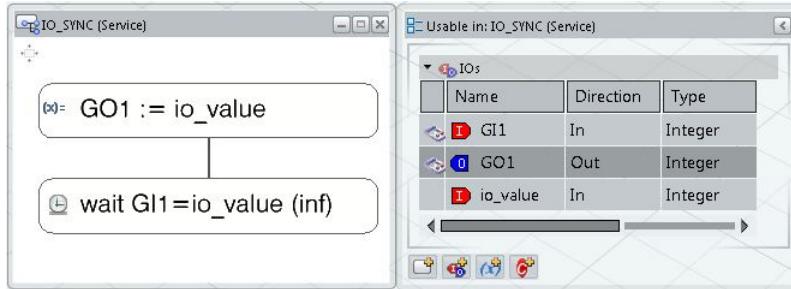
To customize a translator follow these steps:

1. [Determine the DELMIA instruction structure](#)
2. [Determine the AST structure](#)
3. [Make a copy of the translator](#)
4. [Write the class to translate the instruction](#)
5. [Modify the translator to use your class](#)
6. [Update options to use your translator](#)

1. Determine the desired DELMIA instruction structure for CALL IO_SYNC

Lets assume you want IO_SYNC to do the same IO handshaking during simulation as it does on the FANUC controller. We will create a task in DELMIA and then when a CALL IO_SYNC instruction is encountered, we will create an instruction to run the task and pass the input argument.

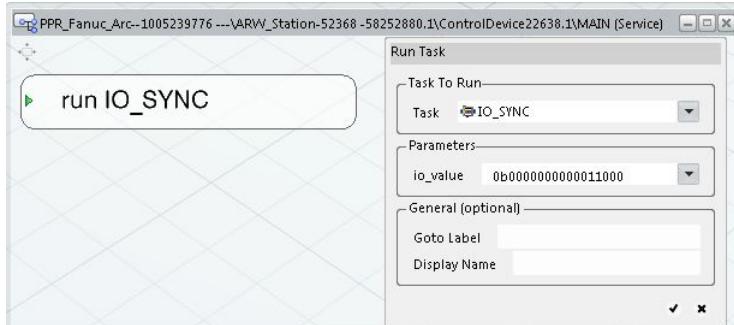
Fig.1: IO_SYNC Task



- IO_SYNC is a Task
- IO_SYNC has 1 integer input (io_value)
- The 1st instruction sets the group output (GO1) value
- The 2nd instruction waits for the group input (GI1) value to be the same.

The instruction to run the task will look like this:

Fig.2: Running IO_SYNC



- The input is changed from a string to an integer in binary notation.

There are many possible equivalent solutions in DELMIA. This is just one possible example.

2. Determine the AST structure for CALL IO_SYNC

The AST structure is determined by the parser. Since the parser cannot be customized, you just need to identify the AST structure that the parser creates for the CALL IO_SYNC instruction. During the upload and download commands, there is an option to export the AST as an XML document. With this option you can observe the AST structure simply by uploading a FANUC TP program that calls IO_SYNC.

NOTE: During translation, AST is **not** an XML document but it can be exported to an XML document.

1. Set the environment variable DNB_OLP_AST_FILE to a file path. The AST will be exported to that XML file.
set DNB_OLP_AST_FILE=C:\tmp\ast.xml
2. Rename MAIN.txt to MAIN.LS, and run the upload command using the DELMIA Fanuc Translator to upload the file MAIN.LS. MAIN.txt can be found in the samples folder listed above.
3. Open the document C:\tmp\ast.xml. Within it, you will find the following XML notation for the CALL IO_SYNC instruction.

```
<Run>
  <Statement id="LineNumber">
    <ConstInteger>1</ConstInteger>
    <Operator>:</Operator>
  <Statement>
    <Format> </Format>
    <Statement id="Call">
      <Keyword>CALL</Keyword>
      <Format> </Format>
      <Identifier>IO_SYNC</Identifier>
      <Statement id="Arguments">
        <Operator>(</Operator>
        <Statement id="Argument">
          <String>'0000000100000000'</String>
        </Statement>
        <Operator>)</Operator>
      </Statement>
    </Format> </Format>
  </Statement>
<Format> </Format>
```

```
<Operator>;</Operator>
<NewLine/>
</Run>
```

3. Create a copy of the translator

We recommend creating your own copy of the macro library for your customizations. This way it is clear whether you are using the DELMIA standard translator or your custom version. Also, as soon as you save a change to a macro library, it is updated in the database so other users in your enterprise will get your changes.

1. Search for the translator in the database. You can use the search string "VSTA: **" to find all VSTA macro libraries.
2. Use the PLM Access->Duplicate command to create a copy of "DELMIA Fanuc Translator R2016x" and rename it "My Fanuc Translator R2016x".
3. Open your copy of the macro library for editing using the Tools->Macros dialog.

4. Write the class to convert between DELMIA Instruction(s) and AST

We recommend putting all your modifications into new classes. This way it is clear what code has been customized and what is part of the DELMIA standard translator. This will also make it simple to move your customization to a new version of the DELMIA translator.

The translators have been written with Object Oriented design. With this approach, your class can inherit from an existing class in the DELMIA translator. You will only need to write code for your specific customization. This will make your development faster because you can make use of existing class methods to do much of the work. This will also mean that if the standard translator is improved, your customized class can also benefit.

In this case we will create a new class MyIO_SYNC which inherits from FanucCallInstruction.

For details on the code used to translate the IO_SYNC instruction, please see [Translating an Instruction](#).

5. Modify the translator to use your class

All translators have a class which identifies the instruction and determines which class to use to translate it. In FANUC translator this class is FanucInstructionList. By creating your own class, MyInstructionList, which inherits from FanucInstructionList, you can identify when an IO_SYNC instruction is being translated and create an object to translate it.

You also need to modify the TranslatorFactory class to return MyInstructionList instead of FanucInstructionList. This is the **only** original translator file that you will ever need to modify.

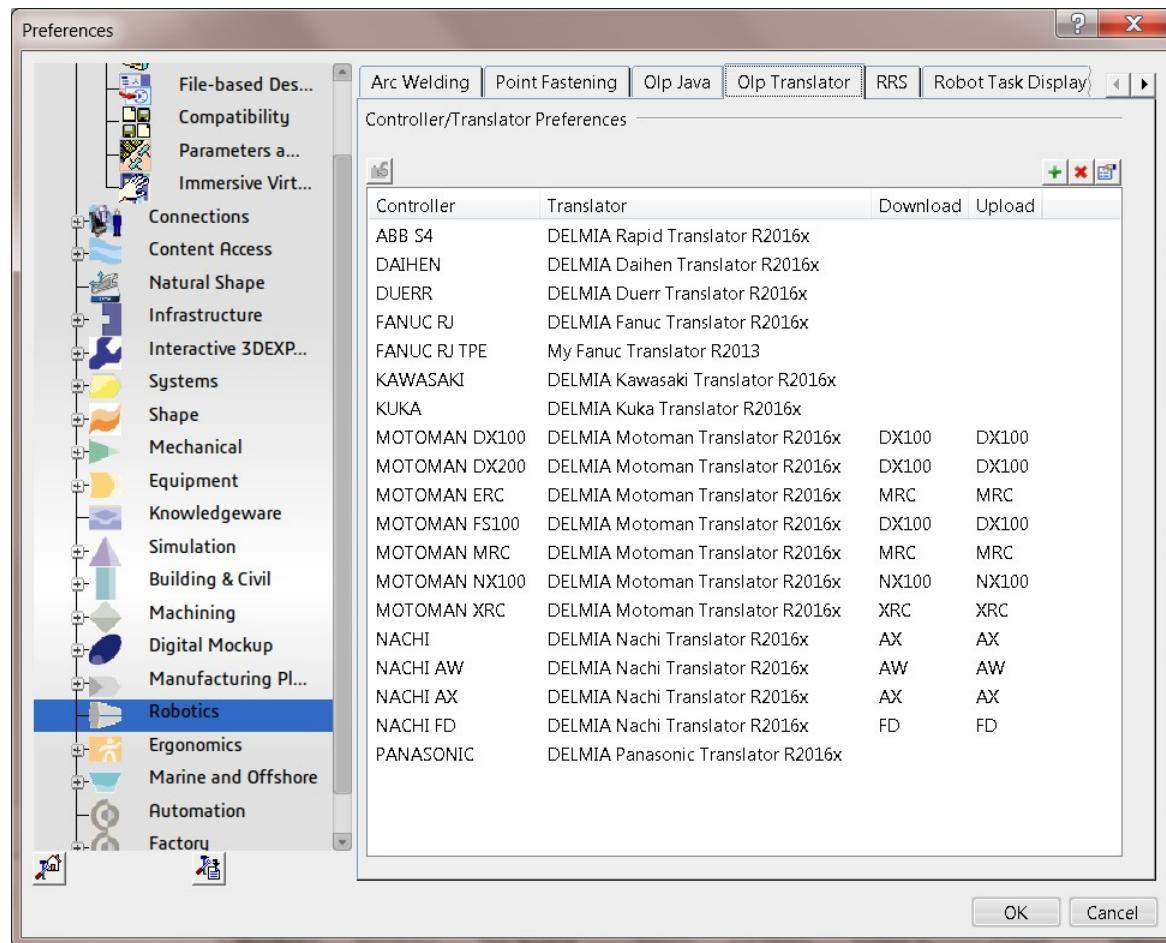
For details on modifying the translator to use your class, please see [Integrating Your Customizations](#).

6. Update Options to use your translator.

Each robot controller type is associated with a translator macro library. You can update this association to use your customized macro library "My Fanuc Translator R2016x".

Depending on the robot manufacturer, there may be more than one controller type listed. In most cases, you will want to update the translator for all the controller types associated with your robot manufacturer. In the case of FANUC there are four entries to update.

Fig.3: OLP Tools->Options dialog.



Translating an Instruction

This example illustrates how to convert a non-standard robot language instruction into DELMIA instructions and back again.

Before you begin:

- You should have read [Robotics Programming Overview](#) and [Customizing a Translator](#).
- You can import a working version of the customized translator from the file `MyFanucTranslatorR2016x.3dxml` supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdOlpofflineProg\samples` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: [MyIO_SYNCSource.htm](#)

This example steps through how to translate a non-standard FANUC instruction. To illustrate the example, we have invented a custom FANUC instruction "IO_SYNC". The details of the IO_SYNC instruction are explained in [Customizing a Translator](#).

```
CALL IO_SYNC('0000000100000000') ;
```

Related Topics
[Robotics](#)
[Programming Overview](#)
[Customizing a Translator](#)
[Integrating Your Customizations](#)

We recommend putting all your modifications into new classes. This way it is clear what code has been customized and what is part of the DELMIA standard translator. This will also make it simple to move your customization to a new version of the DELMIA translator.

The translators have been written with Object Oriented design. With this approach, your class can inherit from an existing class in the DELMIA translator. You will only need to write code for your specific customization. This will make your development faster because you can make use of existing class methods to do much of the work. This will also mean that if the standard translator is improved, your customized class can also benefit.

The modifications required to translate the IO_SYNC instruction are:

- [Determine the Base Class](#)
- [Create the constructor](#)
- [Identify properties and methods to override](#)
- [Create the IO_SYNC task](#)
- [Overriding UploadProcedure](#)
- [Overriding UploadArguments](#)
- [Overriding AppendArguments](#)

1. Determine the Base Class

The first step in object oriented design is to plan the relationships between your classes. In this case we want to create one class to translate FANUC CALL instruction the called task is IO_SYNC. Our class has the following behavior:

- on upload creates an IO_SYNC procedure if it does not already exist
- on upload create a run IO_SYNC instruction from a CALL IO_SYNC instruction
- on upload convert the argument from a string to an integer in binary format
- on download create a CALL IO_SYNC instruction from a run IO_SYNC instruction

- o on download convert the argument from an integer in binary format into a string

Our class takes as input an existing FANUC CALL IO_SYNC instruction (on upload) or DELMIA run IO_SYNC instruction (on download). It does not need to know the instruction type and calling this class when needed is addressed in the use case [Integrating Your Customizations](#).

The behavior of our class is a specialized version of the FanucCallInstruction class which is used to translate a normal CALL instruction into a run instruction. So

```
<CLSCompliant(False)> Public Class MyIO_SYNC
    Inherits FanucCallInstruction
```

2. Create the constructor

Since our class inherits from FanucCallInstruction, we must have a constructor with a similar signature. In this case we simply take the same inputs and pass them

```
Public Sub New(ByRef iTask As FanucTask,
              ByVal iV6Instruction As OlpInstruction, _
              ByRef iAstInstruction As OlpAstBranch)
    MyBase.New(iTask, iV6Instruction, iAstInstruction)
End Sub
```

3. Identify the properties and methods to override

By identifying the differences between the normal CALL translation and the CALL IO_SYNC translation we can identify what behavior we need to override.

Method to Override Reason to Override

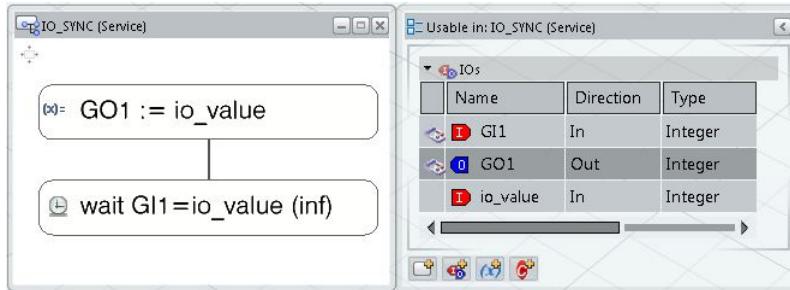
UploadProcedure	The default behavior is to create a blank task if the called task does not exist. Instead, we want to create the IO_SYNC procedure with the
UploadArguments	We need to convert the string argument into an integer and validate that the argument is in the expected format.
AppendArguments	This is used on download to create the AST for the arguments. We need to convert the integer to a string and validate that the argument is

All other behavior is already included in the FanucCallInstruction translator, such as creating the run instruction on upload and creating the CALL instruction AST

4. Creating the IO_SYNC task

First we will write the function to create the task. This function will be called if the task does not already exist. You may recall from [Customizing a Translator](#) that

Fig.1: IO_SYNC Task



We will write a function to create this task. First we create the task and give it a name.

```
Protected Overridable Function CreateIOSYNC() As OlpProcedure
    Dim IO_SYNC As OlpProcedure = Main.TransHelper.Behavior.Procedures.CreateRobotTask()
    IO_SYNC.Name = "IO_SYNC"
    ...

```

Then we add the task input argument.

```
...
Dim io_value As OlpVariable = IO_SYNC.LocalVariables.GetOrCreateIO("io_value",
    DELOlpDataType.delOlpInteger, DELOlpIODirection.delOlpInput)
...

```

Now we need to create the instructions in the task to set the group output and wait for the group input. But before we do this, we need to get the external IOs. We want to use GI[1] as the output and GI[1] as the input.

```
...
Dim ExternalIO As OlpVariables = Main.TransHelper.Behavior.GlobalVariables
Dim go1 As OlpVariable = ExternalIO.GetOrCreateIO("GO1", DELOlpDataType.delOlpInteger, DELOlpIODirection.delOlpOutput)
Dim g1 As OlpVariable = ExternalIO.GetOrCreateIO("GI1", DELOlpDataType.delOlpInteger, DELOlpIODirection.delOlpInput)
...

```

Then we create the set and wait instructions. The last two arguments to CreateInstruction determine where the instructions are created. We want to append the instruction and "True" to create at the end (instead of the beginning).

```
...
Dim Instructions As OlpInstructions = IO_SYNC.Instructions
Dim SetOutput As OlpAssign = Instructions.CreateInstruction(DELOlpInstructionType.delOlpAssign, Nothing, True)
Dim WaitInput As OlpWait = Instructions.CreateInstruction(DELOlpInstructionType.delOlpWait, Nothing, True)
...

```

Next we configure the Set instruction. First, by setting the group output as the destination of the assignment.

```
...
SetOutput.Destination = go1
...

```

Then we configure the value that is assigned to the output. The value can be any expression which is assigned in AST format. DELMIA expressions have a dedicated

Translation. In our case we want a very simple expression which contains just the input argument identifier. This is the AST format (expressed as XML) we want

```
<Expression>
  <Identifier>io_value</Identifier>
</Expression>
```

That AST format is created by the following code and then assigned to the set instruction source.

```
...
Dim SetExpression As OlpAstBranch = Main.TransHelper.AstRoot.CreateBranch(DELOlpAstNodeType.delAstEXPRESSION, "")
SetExpression.AppendLeaf(DELOlpAstNodeType.delAstIDENTIFIER, "", io_value.Name)
SetOutput.Source = SetExpression
...
```

Finally we configure the Wait instruction. The wait expression is also in AST format. In this case we want to wait for the task input argument to be equal to the ex

```
<BinaryExpression>
  <Identifier id="Operand1">G01</Identifier>
  <Operator>=</Operator>
  <Identifier id="Operand2">io_value</Identifier>
</BinaryExpression>
```

That AST format is created with the following code. We also set the WAIT instruction to not timeout.

```
...
WaitInput.UseTimeout = False

Dim WaitExpression As OlpAstBranch = Main.TransHelper.AstRoot.CreateBranch(DELOlpAstNodeType.delAstBINARYEXP, "")
WaitExpression.AppendLeaf(DELOlpAstNodeType.delAstIDENTIFIER, "Operand1", g11.Name)
WaitExpression.AppendLeaf(DELOlpAstNodeType.delAstOPERATOR, "", "=")
WaitExpression.AppendLeaf(DELOlpAstNodeType.delAstIDENTIFIER, "Operand2", io_value.Name)
WaitInput.Condition = WaitExpression

Return IO_SYNC
End Function
```

5. Overriding UploadProcedure

We need to call the CreateIOSYNC function if the IO_SYNC task does not exist. We assume that if it exists that the logic inside the task is correct. You could im~~port~~ is correct the first time you upload a CALL IO_SYNC instruction.

```
Public Overrides Sub UploadProcedure()
  Dim IO_SYNC As OlpProcedure = Nothing

  ' see if the task already exists
  IO_SYNC = Main.TransHelper.Behavior.Procedures.Item("IO_SYNC")

  If IO_SYNC Is Nothing Then
    ' if it does not exist, create it
    IO_SYNC = CreateIOSYNC()
  End If

  ' set the run instruction to call IO_SYNC
  RunInstr.Procedure = IO_SYNC

End Sub
```

6. Overriding UploadArguments

UploadArguments converts the Arguments AST branch into arguments for the DELMIA run instruction. The DELMIA run instruction arguments are also specific FANUC AST format into the expression AST format as described in [OLP Expression Translation](#). Note that an Abstract Syntax Tree (AST) is different for each language.

First we find the FANUC arguments AST branch which is a child of the CALL instruction AST branch. You can find the full AST format for the FANUC CALL ExistsChildByType so that we can detect missing arguments. After finding the argument, we make a copy so we don't modify the parsed FANUC AST.

```
Protected Overrides Sub UploadArguments()
  Dim ArgsNode As OlpAstNode = Nothing
  Dim ExpressionOK As Boolean = False
  If AstInstruction.ExistsChildByType(DELOlpAstNodeType.delAstARGUMENTS, ArgsNode) Then
    Dim ArgsBranch As OlpAstBranch = ArgsNode.Clone
  ...

```

Then we locate the argument. Again we validate that there is only one argument and get the argument with ExistsChildByType in case the CALL IO_SYNC instr

```
...
Dim StringNode As OlpAstNode = Nothing
If ArgsBranch.Count = 1 And ArgsBranch.ExistsChildByType(DELOlpAstNodeType.delAstSTRING, StringNode) Then
  ExpressionOK = True
...
```

We know from the AST format that the argument is an AST leaf node, so now we need to convert it from a string to an integer. We can change the AST node type removing the quotes and adding "0b". Then we update the AST value.

```
...
Dim StringLeaf As OlpAstLeaf = StringNode
StringLeaf.Type = DELOlpAstNodeType.delAstCONST_INTEGER

Dim StringValue As String = StringLeaf.Value
StringLeaf.Value = "0b" & StringValue.Substring(1, StringValue.Length - 2)
...
```

Finally we set the run instruction arguments if they were in the correct format or post a warning if they were not.

```
...
End If
End If
```

```

    If ExpressionOK Then
        RunInstr.Arguments = ArgsBranch
    Else
        Main.ErrorReporter.AddMessage(DEL0lpMessageType.del0lpWarning, "Invalid arguments in call to IO_SYNC. There should be
    End If

End Sub

```

7. Overriding AppendArguments

This is the method used on download to add the arguments to the AST. This is the reverse logic of UploadArguments.

First we get the run instruction arguments. We do not need to make a copy because OlpRun.Arguments returns an AST branch that can be modified.

```

Protected Overrides Sub AppendArguments()
    Dim V6Args As OlpAstBranch = RunInstr.Arguments
    ...

```

Then we verify that there is only one argument, that the argument is an integer, and get that argument.

```

    ...
    Dim ExpressionOK As Boolean = False
    If V6Args.Count = 1 Then
        Dim Arg1 As OlpAstBranch = V6Args.Item(1)
        Dim IntegerNode As OlpAstNode = Nothing
        If Arg1.Count = 1 And Arg1.ExistsChildByType(DEL0lpAstNodeType.delAstCONST_INTEGER, IntegerNode) Then
            ExpressionOK = True
        ...

```

The integer argument is converted to a string. For details on the ConvertIntToString method please see the [full source code](#).

```

    ...
    ConvertIntToString(IntegerNode)
    End If
End If
...

```

Finally, if the arguments were valid, we add the arguments to the instruction AST, if not we post a warning.

```

    ...
    If ExpressionOK Then
        AstInstruction.AppendLeaf(DEL0lpAstNodeType.delAstOPERATOR, "", "(")
        AstInstruction.Append(V6Args)
        AstInstruction.AppendLeaf(DEL0lpAstNodeType.delAstOPERATOR, "", ")")
    Else
        Main.ErrorReporter.AddMessage(DEL0lpMessageType.del0lpWarning, "Invalid arguments in call to IO_SYNC. There should be
    End If

End Sub

```

Integrating Your Customizations

This example illustrates how to integrate into a DELMIA translator a class which translates a custom instruction.

Before you begin:

- You should have read [Robotics Programming Overview](#), [Customizing a Translator](#) and [Translating an Instruction](#).
- You can import a working version of the customized translator from the file `MyFanucTranslatorR2016x.3dxml` supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdolpofflineProg\samples` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: [MyInstructionListSource.htm](#), [TranslatorFactorySource.htm](#)

This example steps through how to identify the type of an instruction and call the appropriate class to translate that instruction. To illustrate the example, we have invented a custom FANUC instruction "IO_SYNC". The details of the IO_SYNC instruction are explained in [Customizing a Translator](#). The class to translate that instruction, `MyIO_SYNC.vb`, is explained in [Translating an Instruction](#).

The modifications required to integrate a custom translator class are:

1. [Create MyInstructionList Class](#)
2. [Override CreateTranslatorForUpload](#)
3. [Override CreateTranslatorForDownload](#)
4. [Modify TranslatorFactory](#)

1. Create MyInstructionList Class

The class `FanucInstructionList` is used to identify the instructions in a FANUC TP program on upload or in a DELMIA task on download. It then creates the appropriate object to perform the translation of the instruction. Therefore, since we need to identify CALL IO_SYNC instructions and create the `MyIO_SYNC` object to translate them, we need to create our own version of this class.

For the same reasons given in [Customizing a Translator](#), we should create our own class, `MyInstructionList`, which inherits from `FanucInstructionList`.

```

<CLSCompliant(False)> Public Class MyInstructionList
    Inherits FanucInstructionList

    Public Sub New(ByRef iTask As FanucTask, ByRef iv6Instructions As OlpInstructions, ByRef iAstInstructions As OlpAstBranch)
        MyBase.New(iTask, iv6Instructions, iAstInstructions)
    End Sub

```

We will need to override 2 methods in this class to create the `MyIO_SYNC` class for upload and download.

2. Override CreateTranslatorForUpload

In the FANUC translator, the method `CreateTranslatorForUpload` is called for each instruction in the FANUC AST. We will override this function to identify CAI IO_SYNC instructions and return a new `MyIO_SYNC` object for them. For all other instructions, we will rely on the base class implementation. After all the objects

Related Topics
[Robotics Programming Overview](#)
[Customizing a Translator](#)
[Translating an Instruction](#)

translate the instructions have been created, the FanucInstructionList class will call Upload on each of them.

If the AST instruction branch is of type delAstRUN then we know that the instruction is a CALL instruction.

```
Protected Overrides Function CreateTranslatorForUpload(ByRef AstInstruction As OlpAstNode) As FanucInstruction
    If AstInstruction.Type = DELOlpAstNodeType.delAstRUN Then
        ...
    End If
```

If it is a CALL, then we need to get the task that is called. Based on the FANUC AST instruction format which was identified in [Customizing a Translator](#), we know the task name is in a node of type delAstIDENTIFIER.

```
...
    Dim AstInstructionBranch As OlpAstBranch = AstInstruction
    Dim TaskName As String =
        AstInstructionBranch.FindChildByType(DELOlpAstNodeType.delAstIDENTIFIER).Value
...

```

If the name of the task is IO_SYNC, then we create a run instruction and a new MyIO_SYNC translator and return the MyIO_SYNC object. The FANUC translator always creates the DELMIA instruction at the same time as the class which will translate it. Other translators may do this differently. You will need to examine the equivalent methods in those translators to see how it is done.

```
...
    If TaskName = "IO_SYNC" Then
        Dim EVTAPIInstruction As OlpInstruction =
            V6Instructions.CreateInstruction(DELOlpInstructionType.delOlpRun, Nothing, True)
        Return New MyIO_SYNC(TaskTranslator, EVTAPIInstruction, AstInstruction)
    End If
...

```

If the instruction is not a CALL MY_IOSYNC, then call the base class implementation of CreateTranslatorForUpload to create the appropriate objects.

```
...
End If
Return MyBase.CreateTranslatorForUpload(AstInstruction)
End Function
```

3. Override CreateTranslatorForDownload

This method identifies the DELMIA instruction and creates the appropriate object to perform the translation. We will override it to identify a run IO_SYNC instruction and return MyIO_SYNC.

Check the instruction type to see if it is a run instruction.

```
Protected Overrides Function CreateTranslatorsForDownload(Val V6Instruction As DELOlp.OlpInstruction) As List(Of FanucInstruction)
    If V6Instruction.Type = DELOlpInstructionType.delOlpRun Then
        ...
    End If
```

Check the called task's name to see if it is IO_SYNC.

```
...
    Dim RunInstruction As OlpRun = V6Instruction
    If RunInstruction.Procedure.Name = "IO_SYNC" Then
        ...
    End If
```

If a run IO_SYNC instruction was found, create the main AST branch for the FANUC instruction and the MyIO_SYNC object. The FANUC translator may use more than 1 object to translate a DELMIA instruction. We only need 1 object, so we need to return a list which has just the MyIO_SYNC object in it. Other translators may create the AST branch at the same time or may not use a list. You will need to examine the equivalent method in other translators to identify what needs to be done.

```
...
    Dim AstEVTAPIInstruction As OlpAstBranch = AstInstructions.CreateBranch(DELOlpAstNodeType.delAstRUN, "")
    Dim Translator As FanucInstruction = New MyIO_SYNC(TaskTranslator, V6Instruction, AstEVTAPIInstruction)
    Dim InstructionTranslators As List(Of FanucInstruction) = New List(Of FanucInstruction)
    InstructionTranslators.Add(Translator)
    Return InstructionTranslators
...

```

If the instruction is not a run IO_SYNC, then call the base class implementation of CreateTranslatorForDownload to create the appropriate objects.

```
...
End If
Return MyBase.CreateTranslatorsForDownload(V6Instruction)
End Function
```

4. Modify TranslatorFactory to create MyInstructionList

The final step in customizing a translator is to replace an existing translator class with one (or more) of your classes. To do this, your class **must** inherit from the class you are replacing.

[TranslatorFactory.vb](#) is the file which enables you do do this substitution. This is the **only** file you will ever need to modify in the DELMIA translator. The same file is used in all translators. When you move your customizations from one version of a DELMIA translator to another, you will only need to copy the files you created, update the TranslatorFactory.

In our example, we only need to replace one class, FanucInstructionList, with our version, MyInstructionList. The only change we made below was to replace "New FanucInstructionList" with "New MyInstructionList".

```
Public Shared Function NewInstructionListTranslator( _
    ByRef iTask As FanucTask,
    ByRef iV6Instructions As OlpInstructions,
    ByRef iAstInstructions As OlpAstBranch) As FanucInstructionList
    Return New MyInstructionList(iTask, iV6Instructions, iAstInstructions)
End Function
```

OLP Profile Matching

Technical Article

Abstract

This article provides details on translating profiles and the methods used to reuse existing profiles while uploading a robot program.

An Introduction to DELMIA Profiles

Robot language instructions have many parameters. In DELMIA Robotics, we have grouped together these parameters into profiles. For example, the [Motion Profile](#) contains all the parameters related to specifying the speed and acceleration of the robot. There are also profiles for specific manufacturing applications such as Spot Welding and Painting.

The grouping of the parameters in DELMIA frequently does not match the groupings in the robot language or the individual parameter may be set on an instruction directly. During upload, these parameters will need to be grouped according to the DELMIA profiles and each unique set of parameter values should have just 1 profile instance created and should be re-used on all the instructions where those parameter values are needed.

When [customizing a translator](#), you have 2 choices for how to create and re-use a profile. The first option is manage the creation and re-used of the profiles directly in the translator. The second option is to assign the parameter values directly to the instruction and specify the rules for when an existing profile should be re-used. Each option will be discussed below.

Managing Profile Creation Directly

This method is similar to how a user creates and assigns profiles through our user interface. Therefore it is a simpler method. However, you may need to write a lot of logic in the translator to get it to work correctly.

Warning: Robot language parameters might not be grouped the same way as the DELMIA profiles.

Warning: Robot program translation is performed sequentially – that is, instruction by instruction.

Given the nature of the problem, some (or a lot) of logic is required to properly manage the creation and retrieval of a profile type given a set of parameters. This logic should only create new profiles if the parameters given do not match any other existent profile, or if the application specifically dictates to do so. It is important to re-use profile as much as possible for performance and usability.

There are 3 steps to working with profiles like this:

1. [Get or Create the profile from the list of profiles](#)
2. [Set the parameter values](#)
3. [Assign the profile to an instruction](#)

Get or Create the profile from the list of profiles

With this method, the profiles are managed by name. You can create a new profile, or retrieve an existing profile with a specific name by calling [GetOrCreateProfile](#) ("MyProfileName") on an object implementing [OlpProfiles](#) by passing the profile name as the argument. Some of those objects can be retrieved like:

```
Dim MyProfiles As OlpProfiles = MainMotionGroup.PrimaryDevice.GetApplicativeProfileList("MyProfileType")
Dim ToolProfiles As OlpProfiles = MainMotionGroup.PrimaryDevice.ToolProfileList
Dim AccuracyProfiles As OlpProfiles = MainMotionGroup.PrimaryDevice.AccuracyProfileList
Dim ObjectProfiles As OlpProfiles = MainMotionGroup.PrimaryDevice.ObjectFrameProfileList
Dim SpotProfiles As OlpProfiles = MainMotionGroup.PrimaryDevice.SpotProfileList
...
...
```

And then the profile is created:

```
Dim MyProfile As OlpProfile = MyProfiles.GetOrCreateProfile("MyProfileName")
```

Set the parameter values

You can then set the parameters of that profile directly by calling [OlpProfile.SetParameter](#).

```
SetParameter(CATBSTR iProfileType,
            CATBSTR iParameterName,
            CATVariant iValue,
            Boolean iMatch)
```

For example to set the attribute "MyParameterName" of the profile we created above to the value 1.0 you would do this:

```
MyProfile.SetParameter("", "MyParameterName", 1.0, False)
```

The first argument is the profile type. When setting a parameter directly on a profile, it is not needed and can be blank. Some profiles can have sub groupings of parameters and this argument is used to set which sub group the parameter is in. The 2nd argument is the parameter name. The 3rd argument is the parameter value and can be any simple data type. The 4th argument is not used when a profile has been retrieved directly from the OlpProfiles list. Its purpose will be discussed in the next section.

The main profile types like Motion Profile, Tool Profile, Spot Profile, etc have dedicated Object types and methods to access their properties. User profiles and Applicative Profiles use the generic object type OlpProfile.

Assign the profile to an instruction

Finally you assign the profile to an instruction. For dedicated profile types you can assign it directly:

```
Dim MotionProfiles As OlpProfiles = MainMotionGroup.PrimaryDevice.MotionProfileList
Dim MyMotionProfile As OlpProfile = MotionProfiles.GetOrCreateProfile("MyMotionProfileName")
MyRobotTarget.MotionProfileOlp = MyMotionProfile
```

For applicative and user profiles you need to assign it by name.

```
MyRobotTarget.SetProfileName("MyProfileType", MyProfile.Name, True)
```

The 1st argument is the type of your profile, the 2nd argument is the name of your profiles and the 3rd argument must be TRUE so that the instruction will go search for a profile that matches the given name.

Using Match Profiles

The main idea with matching profiles is that when translating, parameters are associated to a motion instruction. Rather than writing code to create the profiles and search through created profiles for one that has all the same parameter values, translators can just set the parameters on the instruction and the system will automatically create a new profile or re-use an existing one. This type of profile creation solves the creation/retrieval problem we depicted above.

However, in order for this to work, the system needs to know which parameters to search for and which ones to ignore. This is because many of the parameters of a profile may not specified in the robot program for a particular language.

Setting a parameter

You can set user and applicative profile parameters on a robot motion target or instruction using the SetParameter method. It is the same as the OlpProfile.SetParameter method. Except you need to specify the profile type

```
SetParameter(CATBSTR iProfileType,
            CATBSTR iParameterName,
            CATVariant iValue,
            Boolean iMatch)
```

For example to set the attribute "MyParameter" of a profile to the value 1.0 you would do this:

```
MyRobotTarget.SetParameter("MyProfileType", "MyParameterName", 1.0, True)
```

The first argument is the profile type and must be specified. If a profile with this type does not exist a new user profile with that type will be created. The 2nd argument is the parameter name. If a parameter does not exist on the applicative or user profile, a new user parameter will be added with the datatype of the 3rd argument. The 3rd argument is the parameter value and can be any simple data type. The 4th argument indicates if you want to use this parameter to search for an existing profile to re-use.

For all profiles types that have a method within an OlpController object to retrieve a profile list (Accuracy, Conveyor Tracking, Motion, Object Frame, Paint, Spot and Tool) you can set their parameters in two ways:

Directly on the object profile:

```
MyRobotTarget.MotionProfileOlp.SetTCPSpeedLinear(DEL_OlpMotionProfileUnits.delOlpMotionProfileAbsolute, 0.5, True)
```

Or by retrieving a generic profile and then assigning it to the object:

```
Dim MyMotionProfile As OlpMotionProfile = MotionProfiles.GetOrCreateByMatch(True)
' The True argument in GetOrCreateByMatch is used to avoid duplicate creation of default (empty) profiles
MyMotionProfile.SetTCPSpeedLinear(DEL_OlpMotionProfileUnits.delOlpMotionProfileAbsolute, 0.5, True)
MyRobotTarget.MotionProfileOlp = MyMotionProfile
```

Matching Details

In most cases, you will want to match every property that is set. However, there are some scenarios where this is not desired. There are a few guidelines you must follow in order for matching to work correctly:

1. At least 1 parameter or the profile name for a given profile type must have iMatch set to TRUE. If no parameters are set to match, then the system doesn't know how to find a profile to reuse and a new profile will be created for every instruction and warning will be issued.
2. You can match either the profile name or some parameters, but never both. The profile name is always unique so if you are matching the profile name, it doesn't make sense to also match a parameter.
3. You should match the same parameters on every instruction for a specific profile type. When searching for a matching profile, the 1st profile that uses all the same parameters will be used. If you don't always match the same set of parameters, you may re-use an incorrect profile.

To help debug the mapping behavior you can set the environment variable "DNB_OLP_MATCH_DEBUG" to any value. It will display trace message in the console window describing when a matching profile is found and when a new profile is created.

A Simple Example:

Here is a simple example to illustrate matching. Let's say there is a profile with 3 attributes in addition to the name:

```
MyProfileType
    Attribute1 (String)
    Attribute2 (Integer)
    Attribute3 (Double)
```

Before uploading let say there are 2 profile instances already:

Name	Attribute1	Attribute2	Attribute3
Profile.1	Abc	3	3.14
Profile.2	Xyz	2	2.72

Let's say you are uploading some motions that have 2 of these attributes but attribute 3 is not included in the program.

Instruction Attribute1 Attribute2

Move.1	Abc	1
Move.2	Xyz	2
Move.3	Abc	1

When uploading the 1st move you would make 4 calls:

```
Target1.SetParameter("MyProfileType", "Attribute.1", "Abc", True)
Target1.SetParameter("MyProfileType", "Attribute.2", 1, True)
Target1.SetParameter("MyProfileType", "Attribute.3", 1.1111, False)
Target1.SetProfileName("MyProfileType", "Abc.1", False)
```

The result is a new profile will be created because no existing profile has Attribute1= "Abc" and Attribute2=1. The last 2 calls for setting Attribute.3 and setting the profile name without matching are optional and depend on your desired results. If those arguments were not specified the default name and attribute value would be used for the new profile.

When uploading the 2nd move you would make 4 calls:

```
Target2.SetParameter("MyProfileType", "Attribute.1", "Xyz", True)
Target2.SetParameter("MyProfileType", "Attribute.2", 2, True)
```

```
Target2.SetParameter("MyProfileType", "Attribute.3", 2.2222, False)
Target2.SetProfileName("MyProfileType", "Xyz.2", False)
```

This time the result is that Profile.2 from above will be matched because Attribute.1= “Xyz” and Attribute.2 = 2. The value of attribute 3 does not matter for matching a profile because iMatch is false. Also the name does matter for matching the profile for the same reason. Those value will be set on the profile.

When uploading the 3rd move you would make 4 calls:

```
Target3.SetParameter("MyProfileType", "Attribute.1", "Abc", True)
Target3.SetParameter("MyProfileType", "Attribute.2", 1, True)
Target3.SetParameter("MyProfileType", "Attribute.3", 1.1111, False)
Target3.SetParameter("MyProfileType", "Attribute.4", "Abc.1", False)
```

This time the same profile created for the 1st move will be matched and used again.

So after upload you want to have this:

Name	Attribute1	Attribute2	Attribute3
Profile.1	Abc	3	3.14
Xyz.2*	Xyz	2	2.2222**
Abc.1	Abc	1	1.1111

* The profile name was changed from Profile.2 to Xyz.2 because the name was set without matching.

** The attribute value was changed from 2.72 to 2.2222 because it was set without matching.

An the task would look like this:

Instruction MyProfileType			
Move.1		Abc.1	
Move.2		Xyz.2	
Move.3		Abc.1	

If you do not set the name or an attribute during upload and a new profile is created a default value will be used for the attribute. If an existing profile is matched, the value from the original profile will be kept.

References

- [1] [Robotics Programming Overview](#)
- [2] [Customizing a Translator](#)
- [3] [Translating an Instruction](#)

History

Version: 1 [Jun 2020] Document created

Integrating Drilling and Riveting into your Translator

Use Case

Abstract

This document describes the necessary steps to customize an existing translator so it can translate drill and rivet instructions and their associated profiles during both program upload and download.

Before you Begin

You should have read [Robotics Programming Overview](#), [Customizing a Translator](#) and [Translating an Instruction](#).

You can import a working version of the customized translator from the file “DELMIA XMLCAADrillRivet Translator.3dxml” and a sample dataset supplied in the folder InstallRootFolder\CAADoc\Doc\English\CAAScdOlpOfflineProg\samples\ where `InstallRootFolder` [1] is the folder where the API CD-ROM is installed. Additionally, this folder contains a sample XML program to upload.

On the Customized Translator and the Program Structure

- The XML program structure supplied in the sample file is just for teaching purposes. The Delmia V5/V6 XML format does not specify a structure for this instruction set.
- During the entire upload process, the XML translator does not make use of a parser/scanner. Instead, it iterates through the XML nodes found in an XML document. Thus, the structure of program uploaded is clearly visible and requires no parsing.

Conceptual Overview of Waypoints

Waypoint Targets:

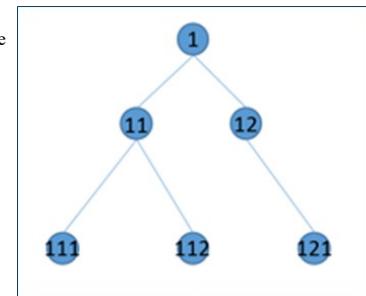
A series of joint or cartesian target locations that define collision free paths the robot can move between. A robot can always move from the parent to a child waypoint and visa-versa. Each parent waypoint target can have multiple possible children to move to. Each process point defines, at least, a waypoint target that it can move to on a collision free path. The hierarchical structure forms an inverted Tree structure with the Root of the Tree having no parent. Each waypoint target must have a WaypointID and a ParentWaypointID assigned. Optionally, the waypoint can have an escape motion type defined.

Waypoint Motion:

A robot motion that contains a waypoint target.

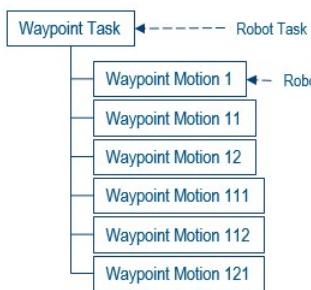
Waypoint Operation:

A sequence instruction containing a series Waypoint motions generated using an assigned Waypoint Task as a reference.



In a Drill/Rivet robot task only one Waypoint Task can be assigned as a reference for its contained Waypoint Operations. The assignment just needs to happen once, in any of the Waypoint Operations instantiated within the Drill/Rivet task.

Waypoint Task:



A task containing only waypoint motions. A waypoint task cannot contain any other type of instruction. Conceptually, a waypoint task can be seen as the reference task that solely defines the hierarchical structure and positional data of waypoint motions. A waypoint task is used as a reference when creating a drill/rivet task, serving as the source of positional data for any waypoint operations instantiated within the drill/rivet task.

In waypoint task image, Waypoint 1 (Root) is the parent of Waypoint 11 and 12. Waypoint 11 is the parent of Waypoint 111 and Waypoint 112 and Waypoint 12 is the parent of Waypoint 121.

Manufacturing Fastener:

An object containing manufacturing positions for the design fasteners/entities (3D points). This object is persisted in the logical behavior Representation and aggregated to Manufacturing cell in the Resource tree. Manufacturing Fastener is used downstream to generate robot tasks.

Manufacturing Pattern – An object used to group Manufacturing Fasteners. This object is persisted in the logical behavior Representation along with Manufacturing Fasteners and aggregated to Manufacturing cell in the Resource tree. Manufacturing Pattern is used downstream to generate robot tasks.

Drill/Rivet Operation:

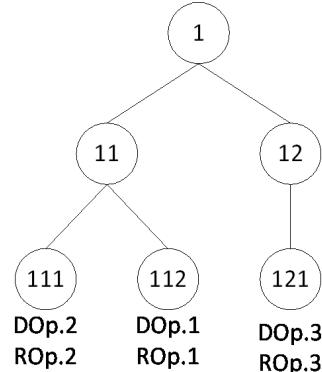
A Robot instruction for drilling/riveting a point. Depending on the content of associated Drill/Rivet Profile, Drill/Rivet Operation would perform different intermediate motions (like Approach, Drill/Rivet Start, Drill/Rivet, Drill/Rivet Complete, Retract) to emulate Drilling/Riveting. Each Drill/Rivet Operation in a Drill/Rivet Task is assigned a parent waypoint, which is used to approach and depart from the operation.

An Example Program

Let us consider a Drill/Rivet task with 4 Drill Operations (DOp.1- ROp.3) and 2 Rivet Operations (ROp.1 and ROp.2). Let us assign Waypoint 111 as parent waypoint of Drill Operation DOp.1 and Rivet Operation ROP.1, Waypoint 112 as parent waypoint of Drill Operation DOp.2 and Rivet Operation ROp.2 and Waypoint 121 as parent waypoint of Drill Operations DOp.3 and ROp.3.

Let's define the order of execution as follows: DOp.1, ROp.1, DOp.2, ROp.2, DOp.3 and finally ROp.3. These are the steps the robot will take to accomplish this:

1. DOp.1 and ROp.1 identify waypoint #112 as its parent. This means we must move to Waypoint #112 in order to execute these operations. Thus, the robot would move to:
 1. Waypoint 1
 2. Waypoint 11
 3. Waypoint 112
2. The robot would perform drill operation DOp.1 and rivet operation ROp.1. Because both share the same parent waypoint, the robot would not perform any waypoint motion between them by default. However, it is possible for the Robot programmer to specify that they wish to force waypoint motion between DOp.1 and ROp.1, even though their parent waypoint is the same. If this option is chosen, the robot would move to waypoint 112 between DOp.1 and ROp.1.
3. DOp.2 and ROp.2 identify waypoint 111 as its parent. In order to reach waypoint 111, the robot would move to:
 1. Waypoint 112, as this is the parent waypoint of the previously executed DOp.1 and ROp.1.
 2. Waypoint 11, as this is the lowest common parent of waypoints 111 and 112
 3. Waypoint 111
4. The robot would perform drill operation DOp.2 and rivet operation ROp.2
5. Drill Operation DOp.3 and Rivet Operation ROp.3 identify waypoint 121 as its parent. In order to reach waypoint 121, the robot would move to:
 1. Waypoint 11
 2. Waypoint 11
 3. Waypoint 1
 4. Waypoint 12
 5. Waypoint 121
6. The robot would perform drill operations DOp.3, and ROp.3



Customizing the XML Translator

To customize the translator, we will follow the steps described in Customizing a Translator:

1. [Determine the DELMIA instruction structure](#)
2. [Determine the XML structure](#)
3. [Make a copy of the translator](#)
4. [Write the required classes to translate the instruction](#)
5. [Modify the translator to use your classes](#)
6. [Update options to use your translator](#)

1. Determine the DELMIA instruction structure

Waypoint Task

To create a waypoint task, we will create a regular robot task and define the waypoint tree structure within by creating Waypoint Motion instructions. A Waypoint Task is just that, a task that contains only waypoint motions. Thus, no special customization is needed on the way the XML translator translates tasks – at least for Waypoint Tasks.

Waypoint Motion

Creating a waypoint motion is not any different from creating any other instruction type, except that they can only be instantiated within a Waypoint Task. Use the CreateInstruction interface on an [OlpInstructions](#) object, and pass in the instruction type – [OlpWaypointMotion](#). Waypoint motion instructions can only be created in a Waypoint Task or under a waypoint Operation. The returned instruction is actually of type [OlpRobotMotion](#), and the main key difference from a regular [OlpRobotMotion](#) is that the waypoint motion target implements [OlpWaypointMotionTarget](#). Thus, we will have to customize the class used to translate Robot Motions (XMLMotionInstruction) in order to translate Waypoint Motions.

Waypoint Operation

To create them, use the CreateInstruction interface on an [OlpInstructions](#) object, and pass in the instruction type – [delOlpWaypointOperation](#). The returned object of type [OlpWaypointOperation](#) contains an Instructions member where we will create the desired waypoint motions. When uploading a waypoint operation, we must set its reference waypoint task. Keep in mind that, when instantiating Waypoint Motion instructions within a Waypoint Operation that has a Waypoint Task set, the only requirement for a fully defined waypoint motion is its waypoint ID. Any additional information set will be overwriting the information stored in the Waypoint Task, so please plan accordingly. As far as our custom translator goes and, since this is a new instruction that did not exist before, we will add a new class that derives from the base Instruction class to translate it.

Drill/Rivet Operation

Creating drill, rivet and drill-rivet operations follows a slightly different approach from what we are accustomed to. This type of operations need to be instantiated from a [OlpTemplate](#). To do so, we will use the CreateInstructionFromTemplate method on the task's [OlpInstructions](#) object with the following parameters:

```
libraryName = "DrillRivet"
templateName = "DrillAction" or "DrillRivetAction" or "RivetAction"
TemplateType = "Spot"
subType = "Action"
RelInstruction is the relative OlpInstruction where to create this action
After is a Boolean value indicating whether to create the instruction before or after the relative instruction
```

We must keep in mind that the return object from this function call is an [OlpTemplate](#), but we can also use the [OlpDrillRivetAction](#) interface to access further properties of this object. The returned object's instructions will contain the drill and/or rivet operations for us to set data on.

The instructions within the returned [OlpTemplate](#) will vary depending on whether we created a Drill, Rivet or a DrillRivet operation. When creating a drill or a rivet action we get a list with a single instruction, as opposed to a DrillRivet action where two instructions are contained. Use the [OlpRobotMotion](#) to modify or access the action's data.

DrillRivet Profile

Drill, Rivet and Drill-Rivet actions must have an associated DrillRivet profile for their definition to be complete. To translate DrillRivet profiles within our custom XML translator, we will derive the XMLController class to add a DrillRivet profile translation method. When instantiating a new DrillRivet profile, we will get a profile that contains slightly different objects depending on the process type we assign it during creation.

Translation Logic

Since we are adding instructions to the translation set, we will have to customize the logic governing the translation process: XMLInstructionList. Additionally, to aid in our translation of waypoint operations, we will need a customized XMLTask.

Others

For the sake of clarity, we will store our Keywords in a separate class from XMLKeywords.

Class	Base Class	Purpose of customization
XMLControllerCustom	XMLController	Translate Drill-Rivet profiles
XMLDrillRivetOperationBaseInstruction	XMLMotionInstructionCustom	Translate Drill and Rivet operations
XMLDrillRivetOperationInstruction	XMLInstruction	Coordinate Drill-Rivet operation upload/download
XMLInstructionListCustom	XMLInstructionList	Control translation logic
XMLKeywordsCustom	XMLKeywords	Storage of keywords
XMLMotionInstructionCustom	XMLMotionInstruction	Translate waypoint instructions
XMLTaskCustom	XMLTask	Aid in WaypointOperation translation
XMLTranslatorCustom	XMLTranslator	Set translator options, fix waypoint operations without waypoint task.
XMLWaypointMotionInstruction	XMLMotionInstructionCustom	Translate WaypointMotions
XMLWaypointOperationInstruction	XMLInstruction	Translate WaypointOperations

2. Determine the XML structure

Note: Have a look at the sample XML program to get a deeper insight into any of the structures outlined below.

Drill-Rivet profiles

Their structure is contained within the Controller element, and is quite simple.

```
<DrillProfileList>
    </DrillProfile>
        Properties
    </DrillProfile>
</DrillProfileList>
<DrillRivetProfileList>
    <DrillRivetProfile>
        Properties
    </DrillRivetProfile>
</DrillRivetProfileList>
```

Waypoint Motion

The Waypoint Motion XML structure is identical to that of a Robot Motion.

```
<Instruction>
    </PersistentParameters>
        <ApplicativeParameters>
            </MotionAttributes>
        </Target>
    </ApplicativeParameters>
</Instruction>
```

Waypoint Operation

The Waypoint Operation structure is quite simple, as it mainly just serves as a Waypoint Motion container. We can identify a waypoint operation by its LateType, which takes a value of DNBWaypointOperationActivity. The link to the Waypoint Task is stored within the PersistentParameters element.

```
<Instruction>
    </PersistentParameters>
```

```
</WaypointOperationInstructions>
</Instruction>
```

Drill, Drill-Rivet and Rivet Operations

The DrillRivet Motion XML structure is identical to that of a Robot Motion. We can identify them by the use of DNBDrlOpActivity, DNBDrlRivetOpActivity and DNBRivetOpActivity.

```
<Instruction>
    </PersistentParameters>
    <ApplicativeParameters>
        </MotionAttributes>
        </Target>
    </ApplicativeParameters>
</Instruction>
```

3. Make a copy of the translator

We recommend creating your own copy of the macro library for your customizations. This way it is clear whether you are using the DELMIA standard translator or your custom version. Also, as soon as you save a change to a macro library, it is updated in the database so other users in your enterprise will get your changes.

1. Search for the translator in the database. You can use the search string "VSTA: *" to find all VSTA macro libraries.
2. Use the Collaborative Tasks->Duplicate command to create a copy of "DELMIA XML R2020x" and rename it to your liking.
3. Open your copy of the macro library for editing using the Tools->Macros dialog.

4. Write the required classes to translate the instruction

Please import the provided XMLCAADrillRivet translator to get details on writing the classes.

5. Modify the translator to use your classes

We will update the TranslatorFactory class to return the appropriate instruction class when instantiating XMLController, XMLMotionInstruction, XMLInstructionList and XMLTask objects. For the newly added classes, we will create new factory methods. Please refer to the provided XMLCAADrillRivet translator for more details.

6. Update options to use your translator

The XML translator supports both V5 and V6 upload and downloads. There is no upload option so there is nothing for us to update there. However, the translator does provide a V5 and a V6 download option. In our case, since we have defined this custom XML structure in a V6-typed XML, we will set the valid download options to "V6", thus dropping "V5" from the supported formats for download.

References

[1] [Robotics Programming Overview](#)

[2] [Customizing a Translator](#)

[3] [Translating an Instruction](#)

History

Version: 1 [Jun 2020] Document created

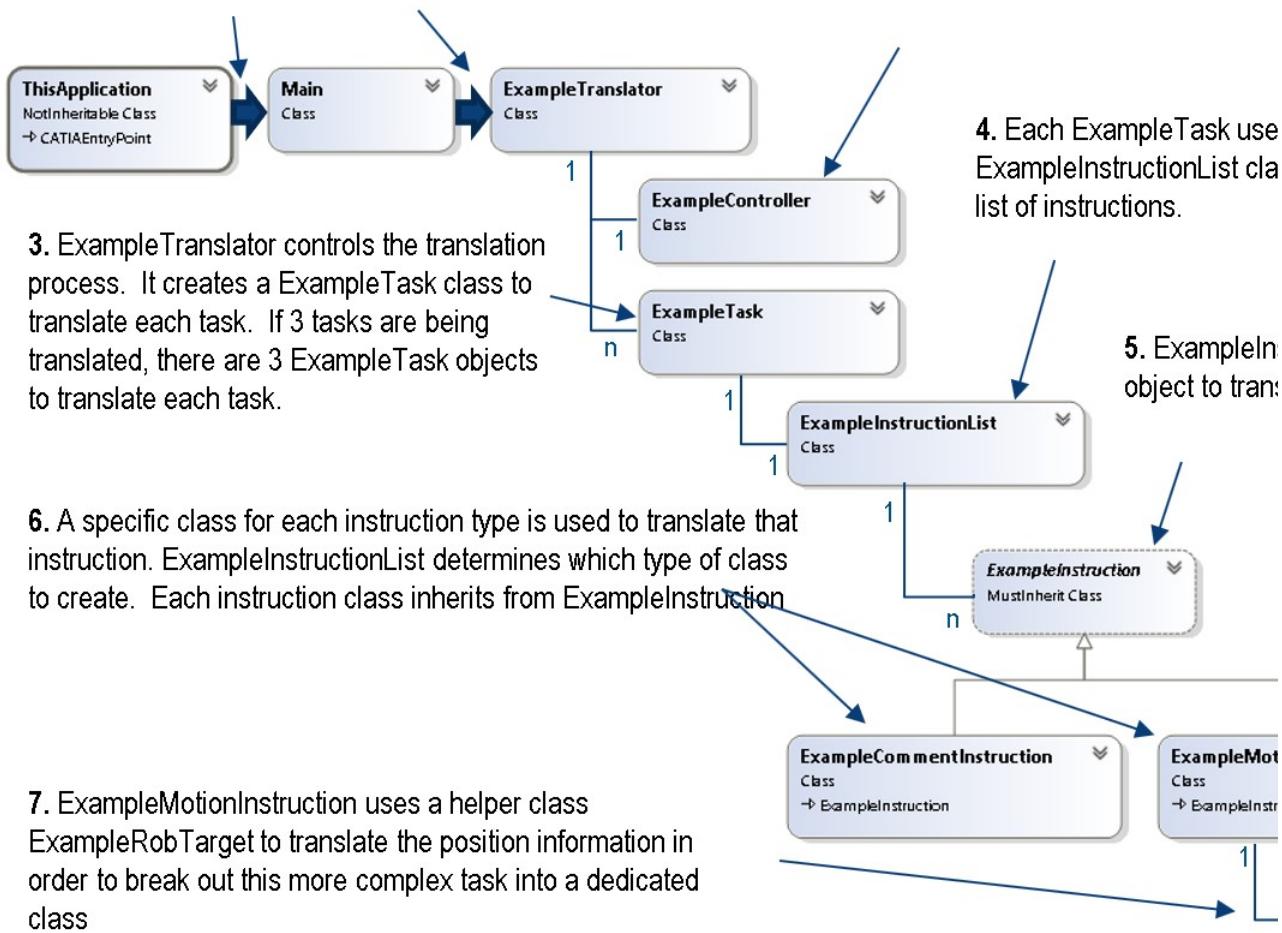
Creating a New Translator

These articles contain further information for creating Translators for Robotics Offline Programming (OLP), as well as an explanation of the provided Example Translator:

1. [Structure of a Translator](#)
2. [Parsing a Robot Program using VB.NET](#)
3. [Translating Instructions](#).
4. [Details of the Example Language](#).

Structure of a Translator

1. Calls are passed through to the primary ExampleTranslator class.



3. ExampleTranslator controls the translation process. It creates a ExampleTask class to translate each task. If 3 tasks are being translated, there are 3 ExampleTask objects to translate each task.

6. A specific class for each instruction type is used to translate that instruction. ExampleInstructionList determines which type of class to create. Each instruction class inherits from ExampleInstruction

7. ExampleMotionInstruction uses a helper class ExampleRobTarget to translate the position information in order to break out this more complex task into a dedicated class

Most 3DEXperience translators are VB.NET programs with associated C++ code. The C++ code is built into 3DEXperience, so this documentation will use the DELMIA Example Translator for reference as it is entirely built in VB.NET. When a translator is opened for editing, there are a few parts available at the base level. ThisApplication.vb includes methods for initiating the translator as an object and launching the appropriate step of translation. Main.vb includes the methods used to initialize and launch the Upload, Download, Parse, and Debug processes. Each of these methods also launches error handling and collects any messages from error handling. These two files sit at the main level of the translator and will call the classes in the Translator directory as the process runs. Some translators additionally have a TranslatorUtils.vb class at this level; this class contains common conversions and utilities used in that translator.

The classes in the Translator directory handle specific functions during the translation process. TranslatorFactory handles generation of new instances of these classes, and should have an entry for any new class added. The [language]Translator.vb file contains the class that Main will call to start Upload or Download for each specific translator. [Language]Keyword.vb contains a series of strings the other functions will use to find relevant items in the incoming program. The Translating Instructions page ([link](#)) contains information on how the other files in this directory are used.

To add more instruction types to the Example Translator, create new classes that inherit from ExampleInstruction.vb and add a method to TranslatorFactory.vb to construct them, then add the appropriate conditions for these instructions to ExampleInstructionList.vb. Most instructions will also need keywords for these instructions added to ExampleKeywords.vb to allow identification of the instruction in the text file and consistency in download to a text file.

Parsing a Robot Program Using VB.NET

Parsing is the process of programmatically reading text and identifying the components of it that make up individual instructions and arguments. In most 3DEXperience translators, this is accomplished by a C++ parser associated with that translator. This parser constructs an Abstract Syntax Tree (AST) out of the items it identifies, which will then be read by the VB.NET upload process for that translator. See Robotics Programming Overview for an explanation of the AST.

The Example Translator is designed to use VB.NET for parse in order to provide a demonstration of how the parse process works. In this translator, ExampleTranslator.Parse() will call ExampleTask.Parse(), similar to the Upload/Download process ([link](#)) in most translators. The ExampleTask.Parse() method checks each line for the keywords that would identify a motion instruction and calls a function to parse the line based on what it finds. If it finds the appropriate keywords, it calls ExampleMotionInstruction.Parse(). If it does not find those keywords, it defaults to calling ExampleCommentInstruction.Parse(). This is the method where additional logic would need to be added to parse non-motion, non-comment lines with the Example Translator, by adding another possibility to the If statement. Note that the terms being checked for are defined in ExampleKeywords.vb; this assures consistency in the keywords used in the code, as well as improving convenience by allowing all instances of each keyword to be changed in one location.

To assist with debugging translators, the TranslatorUtils class contains the method TranslatorUtils.OutputASTForDebug(). This method will output the AST structure as an .xml file. This method expects two arguments - the first is the AST structure to output, an OlpAstBranch variable, and the second is a string, which the method will use to name the output file. You can use the AST XML-file while debugging to identify problems with the parsing logic. For more information on how to debug a translator with breakpoints, see the Simulation | Robotics | Robot Programming | Translator Prerequisites | Debugging Translators section of the User Assistance.

Translating Instructions

Instructions are the individual lines in a 3DExperience simulated robot program. The included translators are intended to use appropriate robot backups and/or programs to create 3DExperience simulation programs, and to use 3DExperience simulation programs to create appropriate robot programs to the translator used.

When Uploading or Downloading, the ExampleTranslator class will find each Task to be translated and pass that task into the ExampleTask class, which uses the passed information to configure an ExampleInstructionList class. ExampleInstructionList contains methods for breaking the task down into a list of instructions and determining the type of each instruction. The Example Translator by default has two classes that could be called from here: ExampleMotionInstruction and ExampleCommentInstruction. Both of these classes inherit from ExampleInstruction to ensure that the same methods can be called from ExampleInstructionList methods. These classes then convert between an AST structure (more information in Parsing a Robot Program) and 3DExperience Tasks and Instructions.

Details of the Example Language

Using the Example Translator

The Example Translator converts between 3DExperience tasks and text files using a custom syntax (the Example Language). The provided Example Translator will translate motion instructions and will render all other instructions as comments. The following is the syntax used in the text files:

- MoveJoint 50% Joint[{0deg,0deg,0deg,0deg,0deg}] Tool[1m,2m,3m,4deg,5deg,6deg] Stop[Fine]
- MoveLine .5m/s Cart[{0.6m,0.2m,0.3m,10deg,20deg,30deg} Config{1}] Tool[.01m,.02m,0.03m,5deg,10deg,15deg] Stop[Fine]

Movement Type:

The Example Language handles Linear and Joint movement. The keywords for these are MoveLine (for linear) and MoveJoint (for joint).

Movement Speed:

The Example Language handles Linear and Relative speeds. Linear (absolute, in 3DExperience movement terms) speeds are measured in meters per second (m/s) by default, while Relative speeds are measured as a percentage of maximum (%).

Target Location:

The Example Language support Joint and Cartesian (Joint and Cart keywords, respectively) position data. By default, it is configured to accept joint positions as such:

- Joint[{10deg, 20deg, 20deg, 0deg, 45deg, 0deg}]

Where "Joint" identifies the type of position data, and each joint's position is then specified in degrees. These values are stored as doubles. The brackets identify the scope of the joint position and the curly braces identify the scope of the joint values.

By default, the Example Language is configured to accept Cartesian positions as such:

- Cart[{.5m, 0m, .1m, 0deg, -90deg, 0deg} Config{1}]

Where "Cart" identifies the type of position data, and the brackets identify the scope of this data. The first set of curly braces identifies the scope of the Cartesian coordinates, measured in meters ("m") and degrees ("deg") and stored as doubles, while "Config" and the second set of curly braces identify the configuration of the robot at this position. Configuration numbers reference the index of the pose saved to the 3DExperience controller, and must be an integer.

Tool Offset:

The key word "Tool" identifies the tool offset specification, contained in its own set of brackets. As with Cartesian positions, this is a Cartesian vector measured in meters (m) and degrees (deg) by default. These values are stored as doubles.

Stop Type:

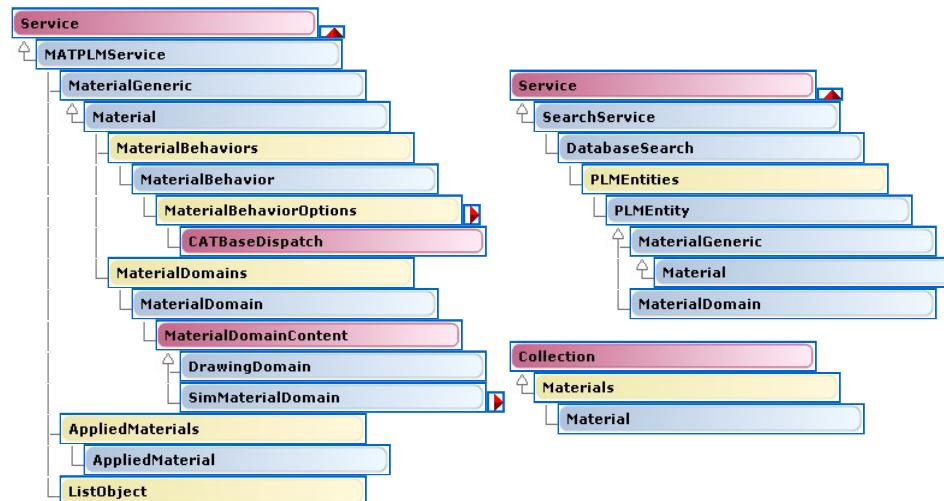
The "Stop" key word identifies the stop type information. The Example Translator provided supports three stop types:

- Stop[Fine] will cause the simulated robot to perform a fine move, coming to a stop at the target location.
- Stop[.05m] will cause the simulated robot to use corner rounding, slowing down as much as is required to come within the specified distance (in meters) of the target location.
- Stop[70%] will cause the robot to brake to 70% of its speed as it approaches the target location before moving on to the next position.

As with the position and tool data, values are stored as doubles.

Material Definition Object Model Map

See Also [Legend](#)



PLMEntity

Use the **GetSessionService** method of the **Application** object to retrieve the **MATPLMServices** object.

VBA Python

```
Dim oMatService As MATPLMServices
Set oMatService = CATIA.GetSessionService("MATPLMServices")

oMatService = CATIA.GetSessionService("MATPLMServices")
```

Using the **MATPLMServices** object, you can:

- Retrieve or set the core material and the covering materials of/to a given object. See [Applying a Material with Domains on a 3D Part](#).
- Remove an applied material.
- Retrieve the materials in the session as a **ListObject** object in which a material is returned as a **PLMEntity** object.
- Create a material. See [Creating a New Material with Domains](#).

From a Material object, you can retrieve two collections:

1. Use the **GetSimulationBehaviors** method to retrieve the **MaterialBehaviors** collection.
2. Use the **GetDomains** method to retrieve the **MaterialDomains** collection.

From a **MaterialDomain** object, use the **MaterialDomainContent** property to retrieve a specific material domain, either a **DrawingDomain** object to create drawing patterns, or a **SimMaterialDomain** object to access behaviors and options of material for simulations.

Creating a New Material with Domains

Before you begin: Note that:

- You should first launch CATIA before executing the macro.
- As macro is demonstrating the "dsc_matref_rep_Sample" domain, some C++ sample of code need to be build before executing the macro. Associated software is located in **CAAMatExtendInterfaces.edu**. An alternative is to adapt the macro to use another domain implementing scripting (as simulation docmain).

Related Topics
[Material Definition](#)
[Object Model Map](#)

Where to find the macro: [CAAScdMatCreateMatSource.htm](#)

This use case can be divided in ten steps:

1. [Retrieve the Material Service](#)
2. [Core Material Creation](#)
3. [Add Domain to Core Material](#)
4. [Covering Material Creation](#)
5. [Add Domain to Covering Material](#)
6. [Retrieve the first domain of a material](#)
7. [Retrieve a domain by discipline](#)
8. [Retrieve a PLM Attribute of a domain](#)
9. [Set a PLM Attribute of a domain](#)
10. [Retrieve Material Domain Content](#)

1. Retrieve the Material Service

```
...
Dim MatService As MATPLMServices
Set MatService = CATIA.GetSessionService("MATPLMServices")
...
```

2. Core Material Creation

```
...
Dim MyMaterialEditor As Editor
Dim MyCoreMatRef As Material

' Creation of Core Material
MatService.PLMCreate "dsc_matref_rep_Core", MyCoreMatRef, MyMaterialEditor
...
```

3. Add Domain to Core Material

For adding a domain to material, you need to get all domains of material, and add in this list your domain.

```
...
Dim myListCoreMatDomains As MaterialDomains
Dim MyDomain As MaterialDomain

MyCoreMatRef.GetDomains myListCoreMatDomains

' Add a Rendering Domain
myListCoreMatDomains.Add "dsc_matref_rep_Rendering", MyDomain

' Add a Composite Domain
myListCoreMatDomains.Add "dsc_matref_rep_Composite", MyDomain

' Add a Drafting Domain
myListCoreMatDomains.Add "dsc_matref_rep_Drafting", MyDomain

' Add a Simulation Domain
myListCoreMatDomains.Add "dsc_matref_rep_SmaOptions", MyDomain
...
```

4. Covering Material Creation

```
...
```

```

Dim MyMaterialEditor As Editor
Dim MyCoveringMatRef As Material

' Creation of Covering Material
MatService.PLMCreat "dsc_matref_ref_Covering", MyCoveringMatRef, MyMaterialEditor
...

```

5. Add Domain to Covering Material

You can only add a Rendering Domain for a Covering Material.

```

...
Dim myListCovMatDomains As MaterialDomains
Dim MyDomain As MaterialDomain

' Retrieve all domains of material
MyCoveringMatRef.GetDomains myListCovMatDomains

' Add a Rendering Domain
myListCovMatDomains.Add "dsc_matref_rep_Rendering", MyDomain
...

```

6. Retrieve the first domain of a material

```

...
' Retrieve all domains of material
Dim myListCovMatDomains As MaterialDomains
MyCoveringMatRef.GetDomains myListCovMatDomains

' From the list, get the first domain of material
Dim MyMatDomain As MaterialDomain
Set MyMatDomain = myListCovMatDomains.Item(1)
...

```

7. Retrieve a domain by discipline

```

...
Dim MyFilteredDomainList 'As ListObject
ReDim MyDomainListDisc(0) 'As Collection
MyDomainListDisc(0) = "dsc_matref_rep_Rendering"

' Get list of domains
Dim MyDomainsList As Object
MyCoreMatRef.GetDomains MyDomainsList

MyDomainsList.GetItemByDiscipline MyDomainListDisc, MyFilteredDomainList
...

```

8. Retrieve a PLM Attribute of a domain

```

...
' Get the V_description of Domain
Dim MyAttrValue As String
MyAttrValue = MyMatDomain.GetAttributeValue("V_description")
...

```

9. Set a PLM Attribute of a domain

```

...
' Set the V_description of Domain
Dim MyAttrValue As String
MyAttrValue = "Modified by User"
MyMatDomain.SetAttributeValue "V_description", MyAttrValue
...

```

10. Retrieve Material Domain Content

```

...
' Retrieve the Material Domain Content
Dim MyMaterialDomainContent As MaterialDomainContent
Set MyMaterialDomainContent = MyMatDomain.MaterialDomainContent
...

```

Creating a New Material with Domains

Before you begin: Note that:

- You should first launch CATIA.
- import the CAAScdFeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdFeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Open any part of the PumpModel
- Launch the structural scenario App and create a structural simulation.

Related Topics
[Material Definition](#)
[Object Model Map](#)

Where to find the macro: [CAAScdMatCreateMatWithPythonSource.htm](#)

This use case can be divided in ten steps:

1. [Retrieve the Material Service](#)
2. [Core Material Creation](#)
3. [Add Domain to Core Material](#)
4. [Covering Material Creation](#)
5. [Add Domain to Covering Material](#)
6. [Retrieve the first domain of a material](#)
7. [Retrieve a domain by discipline](#)
8. [Retrieve a PLM Attribute of a domain](#)
9. [Set a PLM Attribute of a domain](#)

10. [Retrieve Material Domain Content](#)

1. Retrieve the Material Service

```
...
MatService = CATIA.GetSessionService("MATPLMService")
...
```

2. Core Material Creation

```
...
MyCoreMatRef, MyMaterialEditor = MatService.PLMCreat ("dsc_matref_ref_Core", None, None)
...
```

3. Add Domain to Core Material

For adding a domain to material, you need to get all domains of material, and add in this list your domain.

```
...
MyListCoreMatDomains = MyCoreMatRef.GetDomains ()
MyDomain = MyListCoreMatDomains.Add ("dsc_matref_rep_Rendering", None)
MyDomain = MyListCoreMatDomains.Add ("dsc_matref_rep_Composite", None)
MyDomain = MyListCoreMatDomains.Add ("dsc_matref_rep_Drafting", None)
MyDomain = MyListCoreMatDomains.Add ("dsc_matref_rep_SmaOptions", None)
MyDomain = MyListCoreMatDomains.Add ("dsc_matref_rep_Sample", None)
...
```

4. Covering Material Creation

```
...
MyCoveringMatRef, MyMaterialEditor = MatService.PLMCreat ("dsc_matref_ref_Covering", None, None)
...
```

5. Add Domain to Covering Material

You can only add a Rendering Domain for a Covering Material.

```
...
MyListCovMatDomains = MyCoveringMatRef.GetDomains()
MyDomain = MyListCovMatDomains.Add ("dsc_matref_rep_Rendering", None)
...
```

6. Retrieve the first domain of a material

```
...
MyMatDomainsList = iMyCoreMatRef.GetDomains()
MyDomainMat = MyMatDomainsList.Item(1)
...
```

7. Retrieve a domain by discipline

```
...
MyDomainListDisc=[]
MyDomainListDisc.append("dsc_matref_rep_Sample")
MyDomainsList = MyCoreMatRef.GetDomains()
MyFilteredDomainList = MyDomainsList.GetItemByDiscipline (MyDomainListDisc, None)
...
```

8. Retrieve a PLM Attribute of a domain

```
...
MyMatDomName = MyDomainMat.GetAttributeValue("V_description")
...
```

9. Set a PLM Attribute of a domain

```
...
MyMatDomName = "Modified by User"
MyDomainMat.SetAttributeValue ("V_description", MyMatDomName)
...
```

10. [Retrieve Material Domain Content](#)

```
...
MyMaterialDomainContent = MyDomainMat.MaterialDomainContent
...
```

Applying a Material with Domains on a 3D Part

Before you begin: Note that:

- You should first launch CATIA and then import and open the CAAcdMat3DPart.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAcdMat\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Related Topics
[Material Definition](#)
[Object Model Map](#)

Where to find the macro: [CAAcdMatApplyMatSource.htm](#)

This use case can be divided in twelve steps:

- [Open the 3DPart](#)
- [Create the compose link used for the apply material](#)
- [Retrieves the Material Service](#)
- [Material and Domain Creation](#)
- [Apply a Covering and Core Material on Root Occurrence](#)
- [Apply a Covering and Core Material on ReplInstance](#)
- [Apply a Covering and Core Material on Feature](#)

8. [Get Applied-Material on Root Occurrence](#)
9. [Get Applied-Material on RepInstance](#)
10. [Get Applied-Material on Feature](#)
11. [Remove Applied Covering Material](#)
12. [Remove Applied Core Material](#)

1. Retrieves the 3DPart Service

```
...
Dim My3DPartService As PLMNewService
Set My3DPartService = CATIA.GetSessionService("PLMNewService")
...
```

2. Search and Open 3DPart (by its Title, using V_Name attribute)

```
...
oDBSearch.AddEasyCriteria "V_Name", "CAAScdMat3DPart"
oSearchService.Search
Dim MyEditor3DShape As Editor
Dim cPLMEntities As PLMEntities
Set cPLMEntities = oDBSearch.Results

If cPLMEntities.Count <> 0 Then
    Set oPLMEntity = cPLMEntities.Item(1)
    Set oPLMOpenService = CATIA.GetSessionService("PLMOpenService")
    oPLMOpenService.PLMOpen oPLMEntity, MyEditor3DShape
End If
...
```

3. Retrieves the active Object

```
...
Dim MyEditor As Part
Set MyEditor = MyEditor3DShape.ActiveObject
...
```

4. Retrieves the VPMRepReference

```
...
Dim MyVPMRepRef As VPMRepReference
Set MyVPMRepRef = MyEditor.Parent
...
```

5. Retrieves the PartBody

```
...
Dim MyPartBody As Body
Set MyPartBody = MyEditor.MainBody
...
```

6. Retrieves the Reference of 3DPart

```
...
Dim MyRootRef As VPMReference
Set MyRootRef = MyVPMRepRef.Father
...
```

7. Retrieves the RepInstance

```
...
Dim MyRepInstance As VPMRepInstance
Dim ListInstance As VPMRepInstances
Set ListInstance = MyRootRef.RepInstances
Set MyRepInstance = ListInstance.Item(1)
...
```

8. Retrieves the Product Service

```
...
Dim MyProductService As PLMNewService
Set MyProductService = CATIA.GetSessionService("PLMProductService")
...
```

9. Retrieves the Root Occurrence

```
...
Dim MyVPMRootOcc As VPMRootOccurrence
Set MyVPMRootOcc = MyProductService.RootOccurrence
...
```

10. Create the compose link used for the apply material

The apply material can be done on Root Occurrence, RepInstance or PartBody. For each case, you need to compose a link that points a target :

```
...
Dim NoRef As VPMReference
Dim NoRepInst As PLMEntity
Dim NoOcc As VPMOccurrence
...

- Root Occurrence : the RepInstance and Reference have to be NULL

...
Dim myLinkOnRootOcc As AnyObject
Set myLinkOnRootOcc = MyProductService.ComposeLink(MyVPMRootOcc, Nothing, Nothing)
...

- RepInstance : the VPMReference and Reference have to be NULL
```

```

...
Dim myLinkOnRepInst As AnyObject
Set myLinkOnRepInst = MyProductService.ComposeLink(Nothing, MyRepInstance, Nothing)
...

- Feature (PartBody, ...) : the VPMReference has to be NULL, but not the RepInstance.
In automation, the feature is a CATIAResource, you have to not confuse it with VPMReference.

...
Dim myLinkOnFeature As AnyObject
' Create a CATIAResource from Object ( as Body, etc ...)
Set MyReference = MyEditor.CreateReferenceFromObject(MyPartBody)
Set myLinkOnFeature = MyProductService.ComposeLink(Nothing, MyRepInstance, MyReference)
...

```

11. Retrieves the Material Service

```

...
Dim MatService As MATPLMService
Set MatService = CATIA.GetSessionService("MATPLMService")
...

```

12. Material and Domain Creation

The material and domain creation is not described in this page. You can find it in [CAASedMatCreateMat.htm](#).

13. Apply a Covering and Core Material on Root Occurrence

Before that you need to compose your link thanks to Product Service. [here](#)

```

...
Dim MyCoveringAppliedMat As AppliedMaterial
Dim MyCoreAppliedMat As AppliedMaterial

' Apply covering material on Root Occurrence
MatService.SetMaterialCovering myLinkOnRootOcc, MyCoveringMatRef, MyCoveringAppliedMat

' Apply core material on Root Occurrence
MatService.SetMaterialCore myLinkOnRootOcc, MyCoreMatRef, MyCoreAppliedMat
...

```

14. Apply a Covering and Core Material on RepInstance

Before that you need to compose your link thanks to Product Service. [here](#)

```

...
Dim MyCoveringAppliedMat As AppliedMaterial
Dim MyCoreAppliedMat As AppliedMaterial

' Apply covering material on RepInstance
MatService.SetMaterialCovering myLinkOnRepInst, MyCoveringMatRef, MyCoveringAppliedMat

' Apply core material on RepInstance
MatService.SetMaterialCore myLinkOnRepInst, MyCoreMatRef, MyCoreAppliedMat
...

```

15. Apply a Covering and Core Material on Feature

Before that you need to compose your link thanks to Product Service. [here](#)

```

...
Dim MyCoveringAppliedMat As AppliedMaterial
Dim MyCoreAppliedMat As AppliedMaterial

' Apply covering material on PartBody
MatService.SetMaterialCovering myLinkOnFeature, MyCoveringMatRef, MyCoveringAppliedMat

' Apply core material on PartBody
MatService.SetMaterialCore myLinkOnFeature, MyCoreMatRef, MyCoreAppliedMat
...

```

16. Get Applied-Material on Root Occurrence

Before that you need to compose your link thanks to Product Service. [here](#)

```

...
' Materials and Applied-Material for Covering Material
Dim MyCoveringMatRefList As ListObject
Dim MyCoveringAppliedMatList As AppliedMaterials
' Applied-Material for Core Material
Dim MyCoreAppliedMat As AppliedMaterial

' Get Covering Material and Applied-Material on Root Occurrence
MatService.GetMaterialCovering myLinkOnRootOcc, MyCoveringMatRefList, MyCoveringAppliedMatList

' Get Core Material and Applied-Material on Root Occurrence
MatService.GetMaterialCore myLinkOnRootOcc, MyCoreMatRef, MyCoreAppliedMat
...

```

17. Get Applied-Material on RepInstance

Before that you need to compose your link thanks to Product Service. [here](#)

```

...
' Materials and Applied-Material for Covering Material
Dim MyCoveringMatRefList As ListObject
Dim MyCoveringAppliedMatList As AppliedMaterials
' Applied-Material for Core Material
Dim MyCoreAppliedMat As AppliedMaterial

```

```
' Get Covering Material and Applied-Material on RepInstance
MatService.GetMaterialCovering myLinkOnRepInst, MyCoveringMatRefList, MyCoveringAppliedMatList

' Get Core Material and Applied-Material on RepInstance
MatService.GetMaterialCore myLinkOnRepInst, MyCoreMatRef, MyCoreAppliedMat
...
```

18. Get Applied-Material on Feature

Before that you need to compose your link thanks to Product Service. [here](#)

```
...Materials and Applied-Material for Covering Material
Dim MyCoveringMatRefList As ListObject
Dim MyCoveringAppliedMatList As AppliedMaterials
' Applied-Material for Core Material
Dim MyCoreAppliedMat As AppliedMaterial

' Get Covering Material and Applied-Material on Feature
MatService.GetMaterialCovering myLinkOnFeature, MyCoveringMatRefList, MyCoveringAppliedMatList

' Get Core Material and Applied-Material on Feature
MatService.GetMaterialCore myLinkOnFeature, MyCoreMatRef, MyCoreAppliedMat
...
```

19. Remove Applied Covering Material

```
...Materials and Applied-Material for Covering Material
Dim MyCoveringAppliedMat As AppliedMaterial
Dim MyCoveringMatRefList As ListObject
Dim MyCoveringAppliedMatList As AppliedMaterials

' Get Covering Material and Applied-Material on Feature
MatService.GetMaterialCovering myLinkOnFeature, MyCoveringMatRefList, MyCoveringAppliedMatList

' Get the first Applied-Material of list
Set MyCoveringAppliedMat = MyCoveringAppliedMatList.Item(1)

' Remove the applied covering material
MatService.RemoveAppliedMaterial MyCoveringAppliedMat
...
```

20. Remove Applied Core Material

```
...Applied Coe Material and Core Material
Dim MyCoreAppliedMat As AppliedMaterial
Dim MyCoreMatRef As Material

' Get Core Material and Applied Core Material
MatService.GetMaterialCore myLinkOnFeature, MyCoreMatRef, MyCoreAppliedMat

' Remove the applied core material
MatService.RemoveAppliedMaterial MyCoreAppliedMat
...
```

Applying a Material with Domains on a 3D Part

Before you begin: Note that:

- You should first launch CATIA.
- import the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Open any part of the PumpModel
- Launch the structural scenario App and create a structural simulation.

Related Topics
[Material Definition](#)
[Object Model Map](#)

Where to find the macro: [CAAScdMatApplyMatWithPythonSource.htm](#)

This use case can be divided in twelve steps:

1. [Retrieves the simulation, its model and creates the different links](#)
2. [Retrieves the Material Service](#)
3. [Create Covering and Core Material](#)
4. [Apply Covering and Core Material](#)
5. [Retrieve Covering and Core Material](#)
6. [Remove Applied Material](#)

1. Retrieves the simulation, its model and creates the different links

As a first step, the UC retrieves the active editor and retrieves the simulation model.

```
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...
```

Then opens the simulation model and retrieves the root occurrence.

```
oOpenService = CATIA.GetSessionService("PLMOpenService")
oOpenService.PLMOpen (myModel)

myProductEditor = CATIA.ActiveEditor
```

```

MyProductService = myProductEditor.GetService("PLMProductService")

myEntities = MyProductService.EditedContent
myRootProduct = myEntities.Item(1)

myRootOccurrence = MyProductService.RootOccurrence
...

```

The apply material can be done on Root Occurrence, RepInstance or PartBody. For each case, you need to compose a link that points a target.

As next step, the UC compose the needed links

```

...
MyActiveObject = myProductEditor.ActiveObject
MyVPMRepRef = MyActiveObject.Parent
MyPartBody = MyActiveObject.MainBody
MyRootRef = MyVPMRepRef.Father
ListInstance = MyRootRef.RepInstances
MyRepInstance = ListInstance.Item(1)
...

```

Root Occurrence : the RepInstance and Reference have to be NULL

```

...
myLinkOnRootOcc = MyProductService.ComposeLink(myRootOccurrence, None, None)
...

```

RepInstance : the VPMReference and Reference have to be NULL

```

...
myLinkOnRepInst = MyProductService.ComposeLink(None, MyRepInstance, None)
...

```

Feature (PartBody, ...) : the VPMReference has to be NULL, but not the RepInstance.

In automation, the feature is a CATIAResource, you have to not confuse it with VPMReference.

```

...
MyReference = MyActiveObject.CreateReferenceFromObject(MyPartBody)
myLinkOnFeature = MyProductService.ComposeLink(None, MyRepInstance, MyReference)
...

```

2. Retrieves the Material Service

```

...
MyMaterialService = CATIA.GetSessionService("MATPLMService")
...

```

3. Create Covering and Core Material

In this step UC creates a core and covering material

First it Creates a Covering Material

```

...
MyCoveringMatRef, MyMaterialEditor = MyMaterialService.PLMLCreate ("dsc_matref_ref_Covering", None, None)
...

```

Then it Adds Domain to coverin Material

```

...
MyListCovMatDomains = MyCoveringMatRef.GetDomains(None)
MyDomain = MyListCovMatDomains.Add ("dsc_matref_rep_Rendering", None)
...

```

it Creates a core Material

```

...
MyCoreMatRef, MyMaterialEditor = MyMaterialService.PLMLCreate ("dsc_matref_ref_Core", None, None)

```

Then Adds Domain to Core Material

It is described in [CAAScdMatCreateMatWithPython.htm](#).

```

MyListCoreMatDomains = MyCoreMatRef.GetDomains (None)
MyDomain = MyListCoreMatDomains.Add ("dsc_matref_rep_Rendering", None)
MyDomain = MyListCoreMatDomains.Add ("dsc_matref_rep_Composite", None)
MyDomain = MyListCoreMatDomains.Add ("dsc_matref_rep_Drafting", None)
MyDomain = MyListCoreMatDomains.Add ("dsc_matref_rep_SmaOptions", None)
return MyCoveringMatRef, MyCoreMatRef
...

```

4. Apply Covering and Core Material

It first Applies a Core and Covering Material on Root Occurrence

```

...
MyCoveringAppliedMat = MyMaterialService.SetMaterialCovering (myLinkOnRootOcc, iMyCoveringMatRef, None)
MyCoreAppliedMat = MyMaterialService.SetMaterialCore (myLinkOnRootOcc, iMyCoreMatRef, None)
...

```

then Applies a Core and Covering Material on RepInstance

```

...
MyCoveringAppliedMat = MyMaterialService.SetMaterialCovering (myLinkOnRepInst, iMyCoveringMatRef, None)
MyCoreAppliedMat = MyMaterialService.SetMaterialCore (myLinkOnRepInst, iMyCoreMatRef, None)
...

```

Finally Applies a Core and Covering Material on Feature

```
...
MyCoveringAppliedMat = MyMaterialService.SetMaterialCovering (myLinkOnFeature, iMyCoveringMatRef, None)
MyCoreAppliedMat = MyMaterialService.SetMaterialCore (myLinkOnFeature, iMyCoreMatRef, None)
...
```

5. Retrieve Covering and Core Material

First UC retrieves Applied Material on Root Occurrence

```
...
MyCoreMatRef, MyCoreAppliedMat = MyMaterialService.GetMaterialCore (myLinkOnRootOcc, None, None)
MyCoveringMatRefList, MyCoveringAppliedMatList = MyMaterialService.GetMaterialCovering (myLinkOnRootOcc, None, None)
...
```

Then Retrieves Applied Material on RepInstance

```
...
MyCoreMatRef, MyCoreAppliedMat = MyMaterialService.GetMaterialCore (myLinkOnRepInst, None, None)
MyCoveringMatRefList, MyCoveringAppliedMatList = MyMaterialService.GetMaterialCovering (myLinkOnRepInst, None, None)
```

Finally Retrieves Applied Material on Feature

```
...
MyCoreMatRef, MyCoreAppliedMat = MyMaterialService.GetMaterialCore (myLinkOnFeature, None, None)
MyCoveringMatRefList, MyCoveringAppliedMatList = MyMaterialService.GetMaterialCovering (myLinkOnFeature, None, None)
...
```

6. Remove Applied Material

First, UC retrieves Applied Material on Root Occurrence

```
...
MyCoreMatRef, MyCoreAppliedMat = MyMaterialService.GetMaterialCore (myLinkOnRootOcc, None, None)
MyCoveringMatRefList, MyCoveringAppliedMatList = MyMaterialService.GetMaterialCovering (myLinkOnRootOcc, None, None)
...
```

Then, remove Applied Core Material on Root Occurrence

```
...
MyMaterialService.RemoveAppliedMaterial (MyCoreAppliedMat)
...
```

Finally Removes Applied Covering Material on Root Occurrence

```
...
iMaxCnx = MyCoveringAppliedMatList.Count
for index in range(1, iMaxCnx+1):
    MyCoveringAppliedMat = MyCoveringAppliedMatList.Item(index)
    MyMaterialService.RemoveAppliedMaterial (MyCoveringAppliedMat)
...
```

Material Simulation Domain representation

Technical Article

Abstract

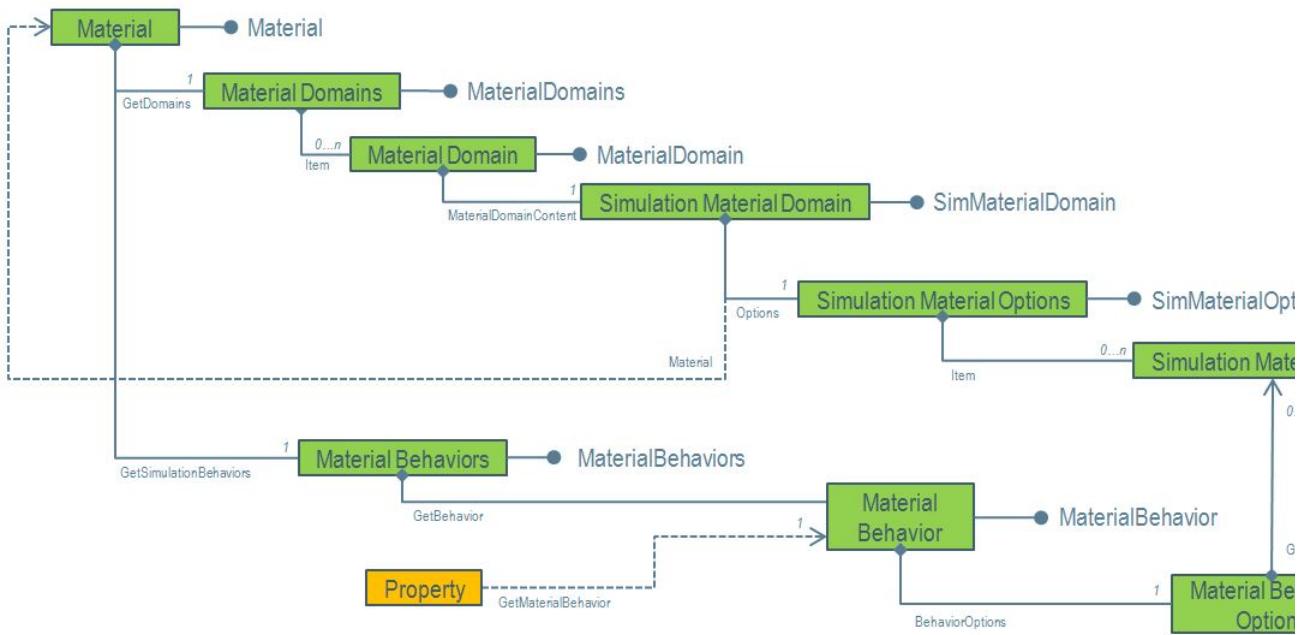
This article's main goal is to help partners understand the data model of the simulation material domain object and identify the APIs necessary to navigate it.

- [Material Simulation Domain](#)

The Material Simulation Domain

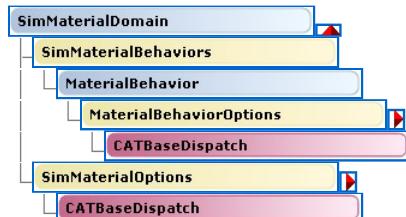
The data model and navigation APIs for the Material Simulation Domain are shown below:

Fig1: Material Simulation Domain navigation graph



SimMaterialDomain Object

See Also [Legend](#)



Use the **MaterialDomainContent** property the **MaterialDomain** object to return a **SimMaterialDomain** object.

VBA Python

```

Dim oMaterialDomain As MaterialDomain
...
Dim oSimMaterialDomain As SimMaterialDomain
Set oSimMaterialDomain = oMaterialDomain.MaterialDomainContent

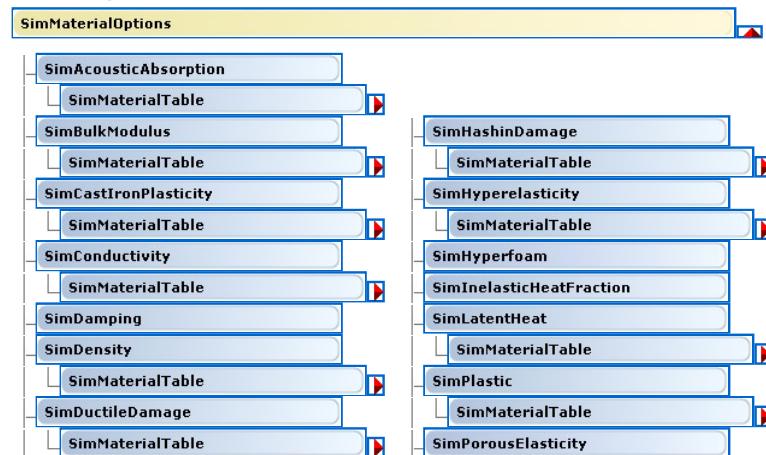
...
oSimMaterialDomain = oMaterialDomain.MaterialDomainContent
...
  
```

From the **SimMaterialDomain** object, use the:

- **Behaviors** property to return the **SimMaterialBehaviors** collection
- **Options** property to return the **SimMaterialOptions** collection. See the [SimMaterialOptions Collection](#) to know which option objects can be created in or retrieved from this collection.

SimMaterialOptions Collection

See Also [Legend](#)





Use the **Options** property the **SimMaterialDomain** object to return a **SimMaterialOptions** collection.

VBA Python

```
Dim oSimMaterialDomain As SimMaterialDomain
...
Dim cSimMaterialOptions As SimMaterialOptions
Set cSimMaterialOptions = oSimMaterialDomain.Options

...
cSimMaterialOptions = oSimMaterialDomain.Options
...
```

Use the **Add** method of the **SimMaterialOptions** collection to create a new simulation material option, such as a **SimAcousticAbsorption** object.

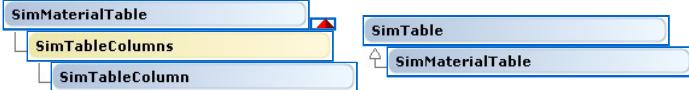
VBA Python

```
Dim oSimAcousticAbsorption As SimAcousticAbsorption  
Set oSimAcousticAbsorption = cSimMaterialOptions.Add("SimAcousticAbsorption")  
  
...  
oSimAcousticAbsorption = cSimMaterialOptions.Add("SimAcousticAbsorption")  
...
```

See [Creation Late Types for Material Features](#) to know how you can create the simulation material options.

SimMaterialTable Object

See Also [Legend](#)



Use the **MaterialTable** property of one of its aggregating object, such as a **SimAcousticAbsorption** object, to return a **SimMaterialTable** object.

VBA Python

```
Dim oSimAcousticAbsorption As SimAcousticAbsorption
...
Dim oSimMaterialTable As SimMaterialTable
Set oSimMaterialTable = oSimAcousticAbsorption.MaterialTable

...
oSimMaterialTable = oSimAcousticAbsorption.MaterialTable
...
```

Creating and Modifying Simulation Material Domain

This use case primarily focuses on the methodology to create or modify a simulation material domain.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: [CAAScdfeaSimulationMaterialDomainSource.htm](#)

This use case can be divided in six steps:

1. [Retrieving the active product](#)
 2. [Creating Material](#)
 3. [Creating Simulation domain](#)
 4. [Creating material option and assign data](#)

5. [Creating Behavior and add options in behavior](#)
6. [Assigning material to part](#)

1. Retrieving the active product

As a first step, the UC retrieves the active product.

```
...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor

Dim myProdService As PLMProductService
Set myProdService = myEditor.GetService("PLMProductService")

Dim myEntities As PLMEntities
Set myEntities = myProdService.EditedContent

Dim myRootProduct As VPMReference
Set myRootProduct = myEntities.Item(1)
...
```

2. Creating Material

In this step UC creates a material reference.

```
...
Dim myMaterialService As MATPLMService
Set myMaterialService = CATIA.GetSessionService("MATPLMService")

Dim myCoreMatRef As Material
Dim myMaterialEditor As Editor
myMaterialService.PLMCreate "dsc_matref_ref_Core", myCoreMatRef, myMaterialEditor
...
```

3. Creating Simulation domain

In this step UC creates simulation domain in the material.

```
...
Dim myDomainList As MaterialDomains
myCoreMatRef.GetDomains myDomainList

Dim myDomain As MaterialDomain
myDomainList.Add "dsc_matref_rep_SmaOptions", myDomain

Dim mySimDomain As SimMaterialDomain
Set mySimDomain = myDomain.MaterialDomainContent
...
```

4. Creating material options and assign data

In this step UC creates new density and elasticity material options.

```
...
Dim myMaterialOptions As SimMaterialOptions
Set myMaterialOptions = MySimDomain.Options

' Creation of a density material option

Dim myDensity As SimDensity
Set myDensity = myMaterialOptions.Add("SimDensity")

Dim myDensityColumn As SimTableColumn
Set myDensityColumn = myDensity.GetMaterialTableColumn(SimDensityDensity)
myDensityColumn.SetCellData 1, 7870

' Creation of an elasticity material option

Dim myElastic As SimElastic
Set myElastic = myMaterialOptions.Add("SimElasticity")

myElastic.ElasticType = SimElasticIsotropic
myElastic.ModuliTimeScaletype = SimElasticLongTerm

Dim myYoungsModulusColumn As SimTableColumn
Set myYoungsModulusColumn = myElastic.GetMaterialTableColumn(SimElasticYoungsModulus)
myYoungsModulusColumn.SetCellData 1, 21100000000.0

Dim myPoissonsRatioColumn As SimTableColumn
Set myPoissonsRatioColumn = myElastic.GetMaterialTableColumn(SimElasticPoissonsRatio)
myPoissonsRatioColumn.SetCellData 1, 0.291
...
```

You can find the available late types for material options creation in the [Creation Late Types for Material Features](#) article.

5. Creating Behavior and add options in behavior

In this step UC creates a material behavior and add options in it.

```
...
Dim myBehaviors As SimMaterialBehaviors
Set myBehaviors = mySimDomain.Behaviors

Dim myBehavior As MaterialBehavior
Set myBehavior = myBehaviors.Add

Dim myBehaviorOptions As MaterialBehaviorOptions
Set myBehaviorOptions = myBehavior.BehaviorOptions

myBehaviorOptions.Add myDensity, 1
myBehaviorOptions.Add myElastic, 2
```

...

6. Assigning material to part

In this step UC assigns the created material to a part.

```

...
' Retrieve the Root Occurrence and the Part Instance
Dim myRootOccurrence As VPMRootOccurrence
Set myRootOccurrence = myProdService.RootOccurrence

Dim myInstances As VPMInstances
Set myInstances = myRootProduct.Instances

Dim myInstance As VPMInstance
Set myInstance = myInstances.Item(1)

Dim myInstanceRef As VPMReference
Set myInstanceRef = myInstance.ReferenceInstanceOf

Dim myRepInstances As VPMRepInstances
Set myRepInstances = myInstanceRef.RepInstances

Dim myPartRepInstance As VPMRepInstance
Set myPartRepInstance = myRepInstances.Item(1)

' Creating link to the part instance

Dim myLinkToPartInstance As AnyObject
Set myLinkToPartInstance = myProdService.ComposeLink(myRootOccurrence, myPartRepInstance, Nothing)

' Assigning material

Dim myCoreAppliedMaterial As AppliedMaterial
myMaterialService.SetMaterialCore myLinkToPartInstance, myCoreMatRef, myCoreAppliedMaterial
...

```

Creating and Modifying Simulation Material Domain

This use case primarily focuses on the methodology to create or modify a simulation material domain.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Launch the structural scenario App and create a structural simulation.

Where to find the macro: [CAAScdfeaSimulationMaterialDomainWithPythonSource.htm](#)

This use case can be divided in six steps:

1. [Retrieves the simulation and its product](#)
2. [Creates Material](#)
3. [Creates Simulation domain](#)
4. [Creates material option and assign data](#)
5. [Creates Behavior and add options in behavior](#)
6. [Assign material to part](#)

1. Retrieves the simulation and its product

As a first step, the UC retrieves the active editor and retrieves the simulation model.

```

...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...

```

Then opens the simulation model and retrieves the root occurrence.

```

...
oOpenService = CATIA.GetSessionService("PLMOpenService")
oOpenService.PLMOpen (myModel)

myProductEditor = CATIA.ActiveEditor
MyProductService = myProductEditor.GetService("PLMProductService")

myEntities = MyProductService.EditedContent
myRootProduct = myEntities.Item(1)

myRootOccurrence = MyProductService.RootOccurrence
...

```

2. Creates Material

In this step UC creates a material reference.

```

...
myMaterialService = CATIA.GetSessionService("MATPLMServices")
myCoreMatRef, myMaterialEditor = myMaterialService.PLMCreat ("dsc_matref_ref_Core", None, None)
...

```

3. Creates Simulation domain

In this step UC creates simulation domain in the material.

```
...
myDomainList = myCoreMatRef.GetDomains (None)
myDomain = myDomainList.Add ("dsc_matref_rep_SmaOptions", None)
mySimDomain = myDomain.MaterialDomainContent
...

```

4. Creates material options and assign data

In this step UC creates new density and elasticity material options.

```
...
myMaterialOptions = mySimDomain.Options

# Creation of a density material option

myDensity = myMaterialOptions.Add("SimDensity")

myDensityColumn = myDensity.GetMaterialTableColumn(win32com.client.constants.SimDensityDensity)

myDensityColumn.SetCellData (1, 7870)

# Creation of an elasticity material option

myElastic = myMaterialOptions.Add("SimElasticity")

myElastic.ElasticType = win32com.client.constants.SimElasticIsotropic
myElastic.ModuliTimeScaletype = win32com.client.constants.SimElasticLongTerm

myYoungsModulusColumn = myElastic.GetMaterialTableColumn(win32com.client.constants.SimElasticYoungsModulus)

myYoungsModulusColumn.SetCellData (1, 211000000000.0)

myPoissonsRatioColumn = myElastic.GetMaterialTableColumn(win32com.client.constants.SimElasticPoissonsRatio)
myPoissonsRatioColumn.SetCellData (1, 0.291)
...

```

You can find the available late types for material options creation in the [Creation Late Types for Material Features](#) article.

5. Creates Behavior and add options in behavior

In this step UC creates a material behavior and add options in it.

```
...
myBehaviors = mySimDomain.Behaviors

myBehavior = myBehaviors.Add()

myBehaviorOptions = myBehavior.BehaviorOptions

myBehaviorOptions.Add (myDensity, 1)
myBehaviorOptions.Add (myElastic, 2)
...

```

6. Assign material to part

In this step UC assigns the created material to a part.

```
...
# Retrieve the Part Instance
myInstances = myRootProduct.Instances

myInstance = myInstances.Item(1)

myInstanceRef = myInstance.ReferenceInstanceOf

myRepInstances = myInstanceRef.RepInstances

myPartRepInstance = myRepInstances.Item(1)

# Creating link to the part instance
myLinkToPartInstance = MyProductService.ComposeLink(myRootOccurrence, myPartRepInstance, None)

# Assigning material
myCoreAppliedMaterial = myMaterialService.SetMaterialCore (myLinkToPartInstance, myCoreMatRef, None)
...

```

Creation Late Types for Material Features

Quick Reference

Abstract

The following reference article documents the different late types that can be used to create material features.

Related Topics
[SimMaterialOptions Collection](#)

- [Material Options Late Types](#)

Material Options Late Types

Material options creation is managed by the **SimMaterialOptions** object by calling the **Add** method from it.

A "late type" must be given to indicate which type of material option is to be created:

Material option	Late type
Isotropic Conductivity	SimConductivity 
Density	SimDensity
Elastic	SimElasticity
Expansion	SimExpansion

Gasket Membrane Elastic	SimGasketMembraneElastic
Gasket Thickness Behavior	SimGasketThicknessBehavior
Gasket Transverse Shear Elastic	SimGasketTransverseShearElastic
Hashin Damage	SimHashinDamage
Plastic	SimPlasticity
Cast Iron Plasticity	SimCastIronPlasticity
Damping	SimDamping
Bulk Modulus	SimBulkModulus
Specific Heat	SimSpecificHeat
Inelastic Heat Fraction	SimInelasticHeatFraction
Latent Heat	SimLatentHeat
Volumetric Drag	SimVolumetricDrag
Acoustic Absorption	SimAcousticAbsorption

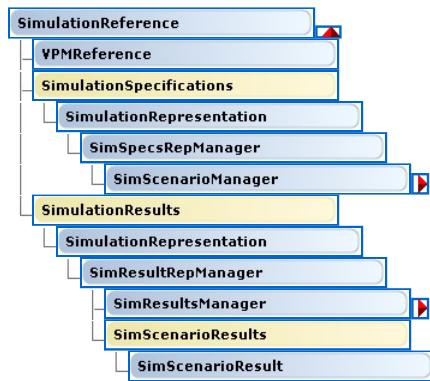
♦ Available with the *RBM - Simulation Scripting* product only.

History

Version: 1 [Jan 2015] Document created

Multiphysics Common Services Object Model Map

See Also [Legend](#)



Use the **SearchService** object to retrieve a simulation from the database, the **PLMOpenService** object to open it, and the **PLMProductService** object to retrieve the root **SimulationReference** object.

VBA Python

```

' Search the database and retrieve a simulation object
Dim oSearchService As SearchService
Set oSearchService = CATIA.GetSessionService("Search")

Dim oDatabaseSearch As DatabaseSearch
Set oDatabaseSearch = oSearchService.DatabaseSearch

oDatabaseSearch.BaseType = "SIMObjSimulationObjectGenericDS"
oDatabaseSearch.AddEasyCriteria "PLM_ExternalID", "TSTSimulation"
oSearchService.Search

Dim cPLMEntities As PLMEntities
Set cPLMEntities = oDatabaseSearch.Results

Dim oPLMEntity As PLMEntity
Set oPLMEntity = cPLMEntities.Item(1)

' Open the simulation object from the database
Dim oOpenService As PLMOpenService
Set oOpenService = CATIA.GetSessionService("PLMOpenService")

Dim oEditor As Editor
oOpenService.PLMopen oPLMEntity, oEditor

' Retrieve the root SimulationReference object
Dim oProdService As PLMProductService
Set oProdService = oEditor.GetService("PLMProductService")

Dim cPLMSimEntities As PLMEntities
Set cPLMSimEntities = oProdService.EditedContent

Dim oSimulationRoot As SimulationReference
Set oSimulationRoot = cPLMSimEntities.Item(1)
  
```

For the Python case, the simulation product is already opened

The code below describe how to get the simulation root reference and how to get the simulation Model

```

...
# Retrieve the active editor
oSimulationEditor = CATIA.ActiveEditor

# Retrieve the root SimulationReference object
oProdService = oSimulationEditor.GetService("PLMProductService")
cPLMSimEntities = oProdService.EditedContent
  
```

```

oSimulationRoot = cPLMSimEntities.Item(1)
# retrieve the simulation model
myModel = oSimulationRoot.Model
...

```

Scenario Representation

Technical Article

Abstract

This article's main goal is to help partners understand the data model of the simulation PLM object and identify the APIs necessary to navigate it.

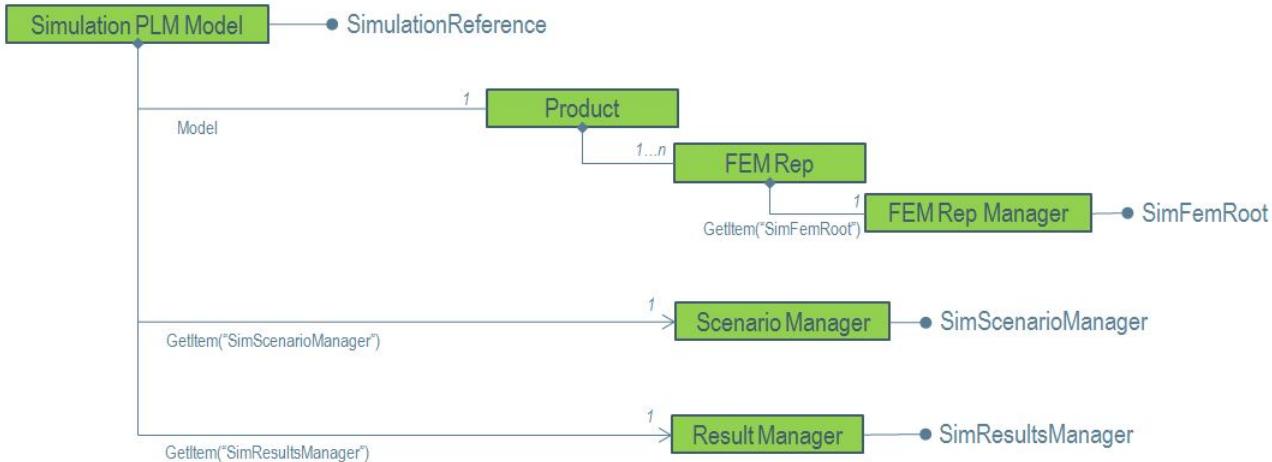
In order to better depict the relationships, the data model and navigation behavior is described through the following five sub-sections:

- [Simulation PLM Object to Managers](#)
- [The FEM Representation Manager](#)
- [The Engineering Connection Manager](#)
- [The Scenario Manager](#)
- [The Results Manager](#)

Simulation PLM Object to Managers

The simulation PLM object is the root feature for any simulation. The following figure describes the data model and navigation APIs from the simulation PLM object to individual managers.

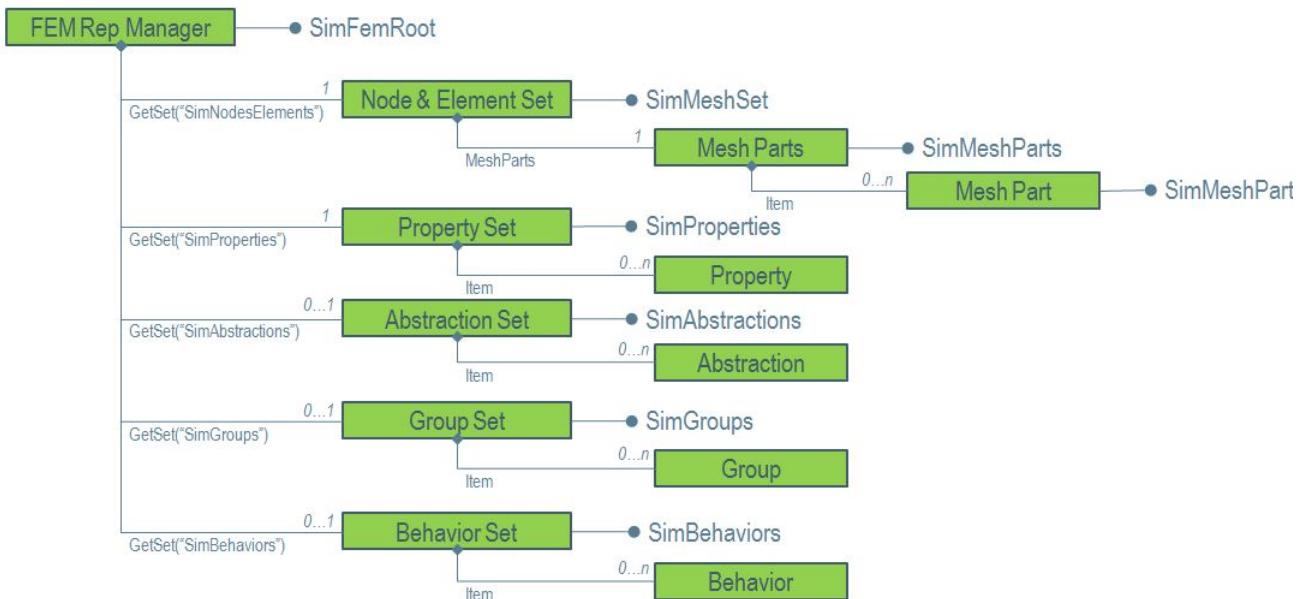
Fig1: Simulation PLM Object to Managers



The FEM Representation Manager

The FEM representation manager contains all the meshes and associated properties necessary to describe the simulation model. The data model and navigation APIs for the FEMRep manager are shown below:

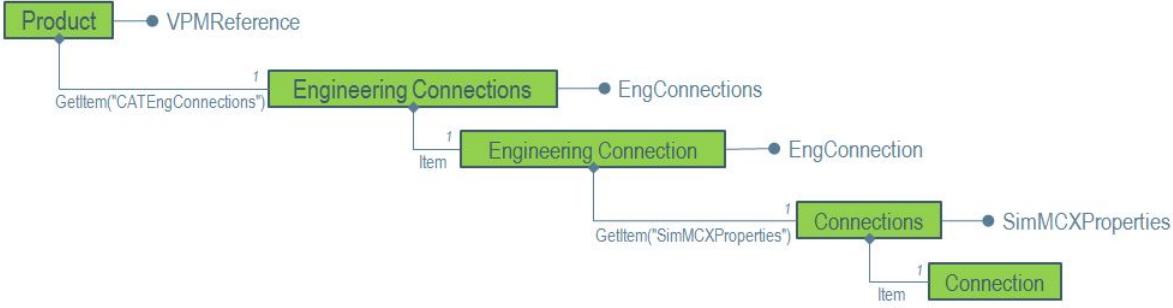
Fig2: FEM Representation Manager to the Model Features



The Engineering Connection Manager

The Engineering connection manager contains all the engineering connections and associated properties necessary to describe the simulation model. The data model and navigation APIs for the Engineering connection manager are shown below:

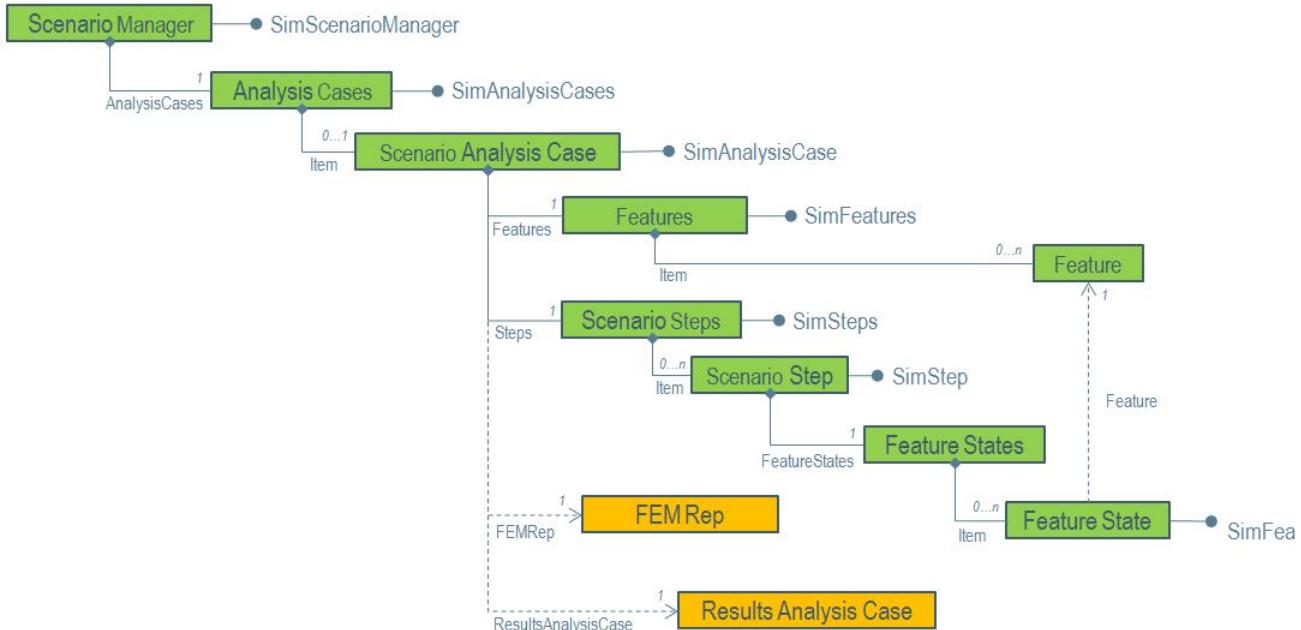
Fig3: Engineering Connection Manager to the Model Features



The Scenario Manager

The scenario manager contains all the pre-processing features necessary to describe the simulation scenario. The data model and navigation APIs for the scenario manager are shown below:

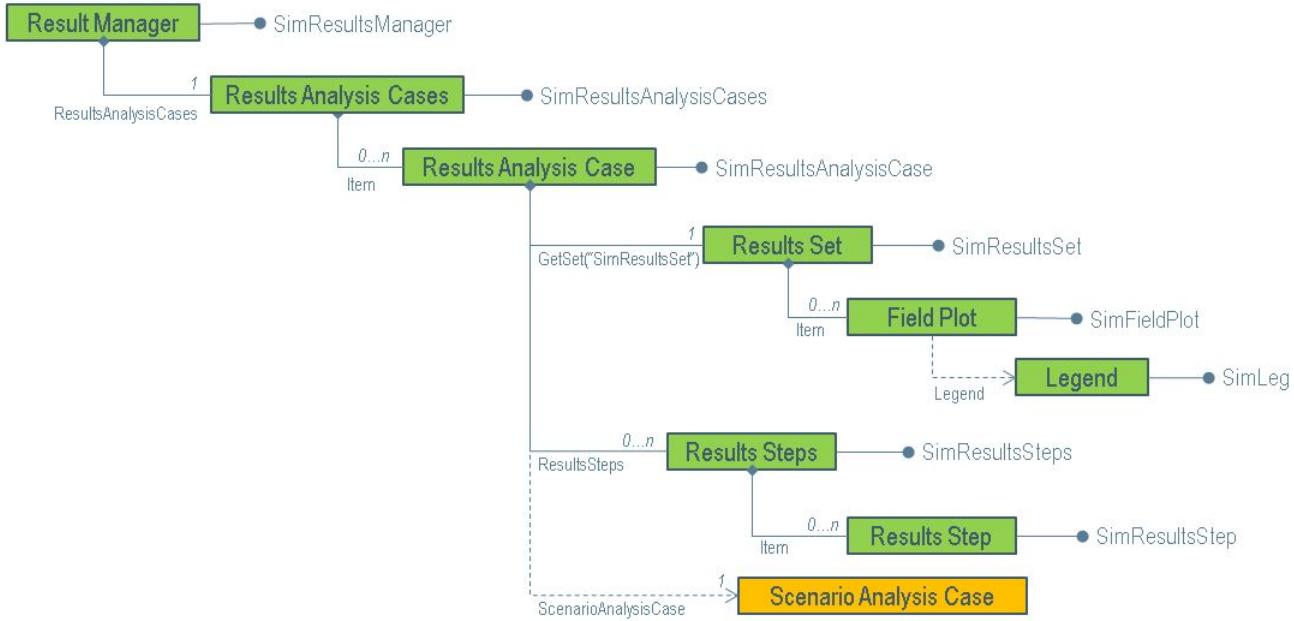
Fig4: Scenario Manager to the Scenario Features



The Results Manager

The results manager contains all the simulation results and the post-processing features necessary to describe the simulation results. The data model and navigation APIs for the results manager is shown below:

Fig5: Results Manager to the Results Features



DS Exporter and Importer options

Technical Article

Abstract

The following reference article describes the different options for the DS importer/exporter and their usage.

This paper explains the options for:

- [The ABAQUS Input File exporter/importer.](#)
 - [The ABAQUS Input File exporter/importer Keys.](#)
 - [The ABAQUS Input File exporter parameters Keys.](#)
 - [The ABAQUS Input File importer parameters Keys.](#)
- [The NASTRAN Bulk Data exporter/importer.](#)
 - [The NASTRAN Bulk Data exporter/importer Keys.](#)
 - [The NASTRAN Bulk Data exporter parameters Keys.](#)
 - [The NASTRAN Bulk Data importer parameters Keys.](#)

ABAQUS Input File exporter/importer

The ABAQUS Input File exporter/importer Keys

Category	Exporter	Importer
Model	<code>Model2Abaqus</code>	<code>Abaqus2Model</code>
Scenario	<code>Scenario2Abaqus</code>	<code>Abaqus2Scenario</code>
Results	<code>Results2Odb</code>	

The ABAQUS Input File exporter parameters Keys

Parameter Keys	Description	Type	Possible options
<code>AllowOverwrite</code>	Allows to overwrite an existing file.	Integer	<code>0 1</code>
<code>CreateLogFile</code>	Create a Log File.	Integer	<code>0 1</code>
<code>LogFolderPath</code>	Log File Path.	String	
<code>FluidFormulation</code>	Changes the mesh element types exported to support appropriate use in an Abaqus input file.	Integer	<code>0 1</code>
<code>PrintPartName</code>	Only for Model2Abaqus exporter. Export the mesh data for each mesh part separately for the "Mesh of Assembly" finite element representation.	Integer	<code>0 1</code>
<code>PrintPartName</code>	Only for Model2Abaqus exporter.	Integer	<code>0 1</code>

The ABAQUS Input File importer parameters Keys

Parameter Keys	Description	Type	Possible options
<code>CreateLogFile</code>	Create a Log File.	Integer	<code>0 1</code>
<code>LogFolderPath</code>	Log File Path.	String	
<code>StructuringOption</code>	None : Generates a mesh part per mesh dimension. Section : Generates a mesh part per property.	String	<code>None Section</code>
<code>ConnectorImport</code>	Import all connector-related objects.	Integer	<code>0 1</code>

NASTRAN Bulk Data exporter/importer

The NASTRAN Bulk Data exporter/importer Keys

Category	Exporter	Importer
Model	<i>Model2Nastran</i>	<i>Nastran2Model</i>
Scenario	<i>Scenario2Nastran</i>	<i>Nastran2Scenario</i>

The NASTRAN Bulk Data exporter parameters Keys

Parameter Keys	Description	Type	Possible options
GranularityGroups	Export groups (Only elements and nodes groups are supported). SET to write the groups before the bulk data in the output file.	Integer	1 0
GroupsLabel	SET1 to write the groups after the bulk data.	String	SET SET1
GranularityMaterials	Export materials.	Integer	1 0
MaterialsStartingID	Starting index for the numbering of the exported materials.	Integer	>=1
GranularityProperties	Export properties.	Integer	1 0
PropertiesStartingID	Starting index for the numbering of the exported properties.	Integer	>=1
GranularitySpecialElements	Element connectivity and specifications are exported according to hard-coded translation.	Integer	1 0
AnalysisCase	Export analysis case.	Integer	1 0
AnalysisCaseSteps	Specify list of steps to be exported.	Array of Integer	
AnalysisCaseGranularityRestraints	Export restraints.	Integer	1 0
AnalysisCaseGranularityLoads	Export loads.	Integer	1 0
AnalysisCaseGranularityOutputs	Export outputs.	Integer	1 0
AnalysisCaseGranularityInitials	Export initial conditions.	Integer	1 0 PARAM,POST,- 1 \$ original OP2
OptionsParamPost	Instruct Nastran about generation of output files.	String	PARAM,POST,-2 \$ general OP2
OptionsExportFormat	Sets the format for bulk data in exported file	String	PARAM,POST,0 \$ XDB
OptionsBeamProperties	YES: Exports beam properties into CBEAM and PBEAM/PBEAML data cards only. NO : Exports beam properties into a mix of BAR and BEAM data cards based on whether it is a Euler-Bernoulli or Timoshenko in Abaqus. CELAS1/PELAS : Exports axial connectors and springs into CELAS1 elements and PELAS properties.	String	Not Enabled Free field Fixed small field Fixed large field YES NO
OptionsAxialSpring	CELAS2: Exports axial connectors and springs into elements without reference to a property entry.	String	CELAS2 CELAS1/PELAS
OptionsComposite	Exports the composite properties to PCOMP or PCOMPG.	String	PCOMP PCOMPG Blank Hill Hoffman
OptionsCompositeFailureTheory	Sets the failure theory for the FT field in PCOMP or PCOMPG.	String	Tsai-wu Maximum Strain
OptionsCompositeStressStrainOutputForPlies	Sets the stress/strain output request field for all the plies in all the PCOMP/PCOMPG cards.	String	NO YES
OptionsCompositeMaximumStrainTheory	This determines Xt, Xc, Yt, Yc, and S fields at MAT8 are stress or strain allowables.	String	STRESS STRAIN
OptionsTruss	Exports the truss elements and associated properties into a CROD/PROD or a CONROD.	String	CROD/PROB CONROD
OptionsBeamConnector	Exports a beam connector to CBUSH/PBUSH or RBE2. If exporting to CBUSH/PBUSH data card, one can specify either translational stiffness or rotational stiffness or both, otherwise export would calculate high stiffness values.	String	CBUSH/PBUSH RBE2
OptionsPBUSHTranslationalStiffness	Specify translational stiffness to be used at CBUSH/PBUSH exporting from beam connectors.	Double	
OptionsPBUSHRotationalStiffness	Specify rotational stiffness to be used at CBUSH/PBUSH exporting from beam connectors.	Double	
OptionsUserDefinePBUSHTranslationalStiffness	Enable export to use user specified translational stiffness to be used at CBUSH/PBUSH exporting from beam connectors.	Int	0 1
OptionsUserDefinePBUSHRotationalStiffness	Enable export to use user specified rotational stiffness to be used at CBUSH/PBUSH exporting from beam connectors.	Int	0 1
OptionsMaximumIOTime	Specifies the maximum run time in CPU minutes.	Double	>=0 (No value)

The NASTRAN Bulk Data importer parameters Keys

Parameter Keys	Description	Type	Possible options
CreateLogFile	Create a Log File.	Integer	0 1
LogFolderPath	Log File Path.	String	
StructuringOption	None : Generates a mesh part per mesh dimension.	String	None Section

Section : Generates a mesh part per property.
 ConnectorImport Import all connector-related objects. Integer 0 | 1

Exporting FE Data

This use case primarily focuses on the methodology to export a FEM representation into a formatted inp file.
 Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdFeaExportFEM.3dxml` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdFeaSimulation\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Then, you should open the "CAAScdFeaExportFEM" physical product. You should also check if all representations are loaded and if the FEM Representation is updated. The export will succeed only if the FEM is updated without errors.

Where to find the macro: [CAAScdFeaExportFEMSource.htm](#)

This use case can be divided in six steps:

1. [Retrieve the active editor](#)
2. [Retrieve the FEM Root](#)
3. [Retrieve the simulation exporter](#)
4. [Retrieve the exporter arguments](#)
5. [Set the export parameters](#)
6. [Execute the export](#)

1. Retrieve the active editor

As a first step, the UC retrieves the active editor.

```
...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor
...
```

2. Retrieve the FEM Root

In the next step, the UC retrieves the `SimFemRoot` object. We need to iterate on the Rep Instances to find the FEM Root object.

```
...
Dim myContext As PLMProductService
Set myContext = myEditor.GetService("PLMProductService")
Dim myFEMRoot As SimFemRoot
For Each myEntity In myContext.EditedContent
    Dim myRef As VPMReference
    Set myRef = myEntity
    Dim myReps As VPMRepInstances
    Set myReps = myRef.RepInstances
    For Each myRep In myReps
        Dim myRepRef As VPMRepReference
        Set myRepRef = myRep.ReferenceInstanceOf
        Dim attr As String
        attr = myRepRef.GetAttributeValue("V_discipline")
        If ( attr = "FEM" ) Then
            Set myFEMRoot = myRep.GetItem("SimFemRoot")
        End If
    Next
Next
If (IsObject(myFEMRoot) = false) Then
    Exit Sub
End If
...
```

3. Retrieve the simulation exporter

In the next step the UC retrieves the FEM representation exporter.

```
...
Dim mySimManagerSIMExport As SimManagerSIMExport
Set mySimManagerSIMExport = myFEMRoot.GetItem("SimManagerSIMExport")
...
```

4. Retrieve the exporter arguments

In this step the UC retrieves the arguments object that allow to define the units, the export path, and some other options.

```
...
Dim mySimExportArgs As SimImportExportArgs
Set mySimExportArgs = mySimManagerSIMExport.Args
...
```

5. Set the export parameters

In this step the UC defines the export path and the export file name.

Then the UC defines some specific parameters to the exporter. Refer to *DS Exporter and Importer options* [1] for further details.

```
...
Dim myExportPath As String
myExportPath=CATIA.SystemService.Environ("TMP") & "\CAAScdFeaExportFEM.inp"
mySimExportArgs.SetPath(myExportPath)

Dim myParameters As SimParameterSet
Set myParameters = mySimExportArgs.Parameters
myParameters.SetIntParameter "AllowOverwrite", 1
myParameters.SetIntParameter "PrintPartName", 1
...
```

6. Execute the export

In this step the UC executes the export. Refer to *DS Exporter and Importer options* [1] to find the keys corresponding to the DS exporter/importer.

```
...
mySimManagerSIMExport.Export("Model2Abaqus")
...
```

References

[1] [DS Exporter and Importer options](#)

Exporting FE Data

This use case primarily focuses on the methodology to export a FEM representation into a formatted inp file.

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdfeaExportFEM.3dxml` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaSimulation\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Then, you should open the "CAAScdfeaExportFEM" physical product. You should also check if all representations are loaded and if the FEM Representation is updated. The export will succeed only if the FEM is updated without errors.
- Launch the structural scenario App and create a structural simulation.

Where to find the macro: [CAAScdfeaExportFEMWithPythonSource.htm](#)

This use case can be divided in six steps:

1. [Retrieves the simulation and its product](#)
2. [Retrieves the FEM Root](#)
3. [Retrieves the simulation exporter](#)
4. [Retrieves the exporter arguments](#)
5. [Sets the export parameters](#)
6. [Executes the export](#)

1. Retrieves the simulation and its product

As a first step, the UC retrieves the active editor and retrieves the simulation model.

```
...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...
```

Then opens the simulation model and retrieves the product service associated to the product editor

```
...
oOpenService = CATIA.GetSessionService("PLMOpenService")
oOpenService.PLMOpen (myModel)

myProductEditor = CATIA.ActiveEditor
MyProductService = myProductEditor.GetService("PLMProductService")

myEntities = MyProductService.EditedContent
...
```

2. Retrieve the FEM Root

In the next step, the UC retrieves the `SimFemRoot` object. We need to iterate on the Rep Instances to find the FEM Root object.

```
...
for myEntity in myEntities:
    myRef = myEntity
    myReps = myRef.RepInstances
    for myRep in myReps:
        myRepRef = myRep.ReferenceInstanceOf
        attr = myRepRef.GetAttributeValue("V_discipline")
        if (attr == "FEM"):
            myFEMRoot = myRepRef.GetItem("SimFemRoot")
...

```

3. Retrieves the simulation exporter

In the next step the UC retrieves the FEM representation exporter.

```
...
mySimManagerSIMExport = myFEMRoot.GetItem("SimManagerSIMExport")
...
```

4. Retrieves the exporter arguments

In this step the UC retrieves the arguments object that allow to define the units, the export path, and some other options.

```
...
mySimExportArgs = mySimManagerSIMExport.Args
...
```

5. Sets the export parameters

In this step the UC defines the export path and the export file name. Then the UC sets a parameter to allow the exporter to overwrite the export file if this one already exists.

```

myExportPath=CATIA.SystemService.Environ("TMP") + "\CAAScdfeaExportFEM.inp"
mySimExportArgs.SetPath(myExportPath)

myParameters = mySimExportArgs.Parameters
myParameters.SetIntParameter ("AllowOverwrite", 1)
...

```

6. Executes the export

In this step the UC executes the export.

```

...
mySimManagerSIMExport.Export ("Model2Abaqus")
...

```

Importing Data into a FEM Rep from an inp File

This use case primarily focuses on the methodology to import data into a FEM representation from a formatted inp file.

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdfeaImportFEM.3dxml` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaSimulation\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- You should also copy into the folder pointed by the "TEMP" environment variable the "CAAScdfeaImportFEM.inp.txt" file supplied in the same folder as the `3dxml` file. **The file extension must be changed to "inp" instead of "txt".**
- Finally, you should open the "CAAScdfeaImportFEM" physical product. You should also check if all representations are loaded.

Where to find the macro: [CAAScdfeaImportFEMSource.htm](#)

This use case can be divided in seven steps:

1. [Retrieve the active editor](#)
2. [Retrieve the FEM Root](#)
3. [Retrieve the simulation importer](#)
4. [Retrieve the importer arguments](#)
5. [Set the import parameters](#)
6. [Manage the import units](#)
7. [Execute the import](#)

1. Retrieve the active editor

As a first step, the UC retrieves the active editor.

```

...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor
...

```

2. Retrieve the FEM Root

In the next step, the UC retrieves the `SimFemRoot` object. We need to iterate on the Rep Instances to find the FEM Root object.

```

...
Dim myContext As PLMProductService
Set myContext = myEditor.GetService("PLMProductService")
Dim myFEMRoot As SimFemRoot
For Each myEntity In myContext.EditedContent
    Dim myRef As VPMReference
    Set myRef = myEntity
    Dim myReps As VPMRepInstances
    Set myReps = myRef.RepInstances
    For Each myRep In myReps
        Dim myRepRef As VPMRepReference
        Set myRepRef = myRep.ReferenceInstanceOf
        Dim attr As String
        attr = myRepRef.GetAttributeValue("V_discipline")
        If ( attr = "FEM" ) Then
            Set myFEMRoot = myRep.GetItem("SimFemRoot")
        End If
    Next
    If (IsObject(myFEMRoot) = false) Then
        Exit Sub
    End If
...

```

3. Retrieve the simulation importer

In the next step the UC retrieves the FEM representation importer.

```

...
Dim mySimManagerSIMImport As SimManagerSIMImport
Set mySimManagerSIMImport = myFEMRoot.GetItem("SimManagerSIMImport")
...

```

4. Retrieve the importer arguments

In this step the UC retrieves the arguments object that allow to define the units, the export path, and some other options.

```

...
Dim mySimImportArgs As SimImportExportArgs
Set mySimImportArgs = mySimManagerSIMImport.Args
...

```

5. Set the import parameters

In this step the UC defines the import file path.

Then the UC can defines some specific parameters to the importer. Refer to *DS Exporter and Importer options* [1] for further details.

```
...
Dim myImportFilePath As String
myImportFilePath=CATIA.SystemService.Environ("TMP") & "\CAAScdfeaImportFEM.inp"
mySimImportArgs.SetPath(myImportFilePath)
...
```

6. Manage the import units

In this step the UC describes the units used into the input file.

```
...
Dim MyMagnitude(0)
Dim MyUnits(0)
MyMagnitude(0) = "LENGTH"
MyUnits(0) = "mm"
mySimImportArgs.SetUnits MyMagnitude, MyUnits
...
```

7. Execute the import

In this step the UC executes the import. Refer to *DS Exporter and Importer options* [1] to find the keys corresponding to the DS exporter/importer.

```
...
mySimManagerSIMImport.Import ("Abaqus2Model")
...
```

References

[1] [DS Exporter and Importer options](#)

Importing Data into a FEM Rep from an inp File

This use case primarily focuses on the methodology to import data into a FEM representation from a formatted inp file.

Before you begin: Note that:

- You should first launch CATIA and import the CAAScdfeaImportFEM.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaSimulation\samples\`, where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- You should also copy into the folder pointed by the "TEMP" environment variable the "CAAScdfeaImportFEM.inp.txt" file supplied in the same folder as the 3dxml file. **The file extension must be changed to "inp" instead of "txt".**
- You should open the "CAAScdfeaImportFEM" physical product. You should also check if all representations are loaded.
- Launch the structural scenario App and create a structural simulation.

Where to find the macro: [CAAScdfeaImportFEMWithPythonSource.htm](#)

This use case can be divided in seven steps:

1. [Retrieves the simulation and its product](#)
2. [Retrieves the FEM Root](#)
3. [Retrieves the simulation importer](#)
4. [Retrieves the importer arguments](#)
5. [Sets the import parameters](#)
6. [Manage the import units](#)
7. [Execute the import](#)

1. Retrieves the simulation and its product

As a first step, the UC retrieves the active editor and retrieves the simulation model.

```
...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...
```

Then opens the simulation model and retrieves the product service associated to the product editor

```
...
oOpenService = CATIA.GetSessionService("PLMOpenService")
oOpenService.PLMOpen (myModel)

myProductEditor = CATIA.ActiveEditor
MyProductService = myProductEditor.GetService("PLMProductService")

myEntities = MyProductService.EditedContent
...
```

2. Retrieves the FEM Root

In the next step, the UC retrieves the `SimFemRoot` object. We need to iterate on the Rep Instances to find the FEM Root object.

```
...
for myEntity in myEntities:
    myRef = myEntity
    myReps = myRef.RepInstances
    for myRep in myReps:
        myRepRef = myRep.ReferenceInstanceOf
        attr = myRepRef.GetAttributeValue("V_discipline")
```

```

if (attr == "FEM"):
    myFEMRoot = myRepRef.GetItem("SimFemRoot")
...

```

3. Retrieves the simulation importer

In the next step the UC retrieves the FEM representation importer.

```

...
mySimManagerSIMImport = myFEMRoot.GetItem("SimManagerSIMImport")
...

```

4. Retrieves the importer arguments

In this step the UC retrieves the arguments object that allow to define the units, the export path, and some other options.

```

...
mySimImportArgs = mySimManagerSIMImport.Args
...

```

5. Sets the import parameters

In this step the UC defines the folder path where the importer can find the input file. Then the UC sets a parameter to allow the exporter to overwrite the export file if this one already exists.

```

...
myImportFilePath=CATIA.SystemService.Environ("TMP") + "\CAAScdfeaImportFEM.inp"
mySimImportArgs.SetPath(myImportFilePath)
...

```

6. Manage the import units

In this step the UC describes the units used into the input file.

```

...
MyMagnitude = []
MyUnits = []

MyMagnitude.append("LENGTH")
MyUnits.append("mm")
mySimImportArgs.SetUnits (MyMagnitude, MyUnits)
...

```

7. Executes the import

In this step the UC executes the import.

```

...
mySimManagerSIMImport.Import ("Abaqus2Model")
...

```

Managing Simulation Feature Support

This article explains how to define and manage simulation feature support.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Make sure that you have all of the licenses required for launching execution and perform a solve operation in the interactive session for a similar simulation to confirm the licenses.

Where to find the macro: [CAAScdfeaLinkAccessSource.htm](#)

This use case can be divided into seven steps:

1. [Retrieving the active Product](#)
2. [Creating the FEM Rep and Retrieving its Rep Manager](#)
3. [Creating Sections](#)
4. [Setting the Meshed Bolt support from Publication](#)
5. [Setting the Shell Section supports from Product structure](#)
6. [Replacing the first Shell Section support](#)
7. [Removing all supports of the Shell Section](#)

1. Retrieving the active Product

As a first step, the UC retrieves the active product.

```

...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor

Dim myProdService As PLMProductService
Set myProdService = myEditor.GetService("PLMProductService")

Dim myEntities As PLMEntities
Set myEntities = myProdService.EditedContent

Dim myRootProduct As VPMReference
Set myRootProduct = myEntities.Item(1)
...

```

2. Creating the FEM Rep and Retrieving its Rep Manager

In this step UC creates a finite element representation model on the product that was just opened.

For further information about creating the FEM Rep and retrieving its Rep Manager, refer to the article [Creating FEM representation](#).

3. Creating sections

In this step UC creates a shell section and a meshed bolt on the part.

For further information about the creation and management of properties, refer to the article [Managing FEM section features](#).

4. Setting the Meshed Bolt support from the Publication

In this step UC sets the meshed bolt support from the publication.

Note that setting support from publication does not require creations links. You can use the publication directly as an argument for the AddLink method.

```
...
'Retrieving the publication

Dim myPublications As VPMPublications
Set myPublications = myModel.Publications

Dim mySupport As VPMPublication
Set mySupport = myPublications.GetItem("Bolt-1")

'Setting the meshed bolt support

Dim myLinkAccess As SimLinkAccess
Set myLinkAccess = myMeshedBolt.GetItem("SimLinkAccess")

    myLinkAccess.AddLink "MainSupport", mySupport
...

```

Important : Note that the "MainSupport" string is a default value. For connection features you will have to use "PRD1", "PRD2", etc... to set the different supports and "HandlerID" or "HandlerID" for fasteners placements.

5. Setting the shell section supports from the product structure

In this step UC sets the shell section supports from the product structure. It first retrieves usefull data to compose the link to the support and then adds this created link to the feature.

```
...
'Retrieving the ws3_bolt_part.2 part occurrence

Dim myRootOccurrence As VPMRootOccurrence
Set myRootOccurrence = myProdService.RootOccurrence

Dim myPartOccurrence As PLMOccurrence
For i = 1 to myRootOccurrence.Occurrences.Count
    If myRootOccurrence.Occurrences.Item(i).Name = "ws3_bolt_part.2" Then
        Set myPartOccurrence = myRootOccurrence.Occurrences.Item(i)
    End If
Next

'Retrieving the ws3_bolt_part.2 part rep instance

Dim myPartInstance As VPMInstance
Set myPartInstance = myPartOccurrence.InstanceOccurrenceOf

Dim myPartInstanceRef As VPMReference
Set myPartInstanceRef = myPartInstance.ReferenceInstanceOf

Dim myPartRepInstance As VPMRepInstance
Set myPartRepInstance = myPartInstanceRef.RepInstances.Item(1)

'Retrieving the ws3_bolt_part.2 part body

Dim myPartRepInstanceRef As VPMRepReference
Set myPartRepInstanceRef = myPartRepInstance.ReferenceInstanceOf

Dim myPart As Part
Set myPart = myPartRepInstanceRef.GetItem("Part")

Dim myPartBody As Body
Set myPartBody = myPart.MainBody

'Composing link to the ws3_bolt_part.2 part body

Dim myReference As Reference
Set myReference = myPart.CreateReferenceFromObject(myPartBody)

Dim myLinkToFirstSupport As AnyObject
Set myLinkToFirstSupport = myProdService.ComposeLink(myPartOccurrence, myPartRepInstance, myReference)

'Setting the shell section first support

Set myLinkAccess = myShellSection.GetItem("SimLinkAccess")

myLinkAccess.AddLink "MainSupport", myLinkToFirstSupport
...
```

6. Replacing the first Shell Section support

In this step UC replaces the first shell section support.

```
...
'Retrieving the publication

Dim myNewSupport As VPMPublication
Set myNewSupport = myPublications.GetItem("Pipe")

'Removing first support
```

```

myLinkAccess.RemoveLink "MainSupport", myLinkToFirstSupport
'Setting the new support
myLinkAccess.AddLink "MainSupport", myNewSupport
...

```

7. Removing all supports of the Shell Section

In this step UC removes all supports of the shell section.

```

...
myLinkAccess.RemoveAllLinks "MainSupport"
...

```

Managing Simulation Feature Support

This article explains how to define and manage simulation feature support.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\DocEnglish\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Launch the structural scenario App and create a structural simulation.
- Make sure that you have all of the licenses required for launching execution and perform a solve operation in the interactive session for a similar simulation to confirm the licenses.

Where to find the macro: [CAAScdfeaLinkAccessWithPythonSource.htm](#)

This use case can be divided into seven steps:

- [Retrieves the simulation and its product](#)
- [Creates the FEM Rep and retrieves its Rep Manager](#)
- [Creates sections](#)
- [Sets the meshed bolt support from publication](#)
- [Sets the shell section supports from product structure](#)
- [Replaces the first shell section support](#)
- [Removes all supports of the shell section](#)

1. Retrieves the simulation and its product

As a first step, the UC retrieves the active editor and retrieves the simulation model.

```

...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...

```

Then opens the simulation model and retrieves the root assembly

```

...
oOpenService = CATIA.GetSessionService("PLMOpenService")
oOpenService.PLMOpen (myModel)

myProductEditor = CATIA.ActiveEditor
MyProductService = myProductEditor.GetService("PLMProductService")

myEntities = MyProductService.EditedContent
myAssembly = myEntities.Item(1)
...

```

2. Creates the FEM Rep and retrieves its Rep Manager

In this step UC creates a finite element representation model on the product that was just opened.

For further information about creating the FEM Rep and retrieving its Rep Manager, refer to the article [Creating FEM representation](#).

3. Creates sections

In this step UC creates a shell section and a meshed bolt on the part.

For further information about the creation and management of properties, refer to the article [Managing FEM section features](#).

4. Sets the meshed bolt support from publication

In this step UC sets the meshed bolt support from the publication.

Note that setting support from publication does not require creation links. You can use the publication directly as an argument for the AddLink method.

```

...
# Retrieves the publication
myPublications = myAssembly.Publications
mySupport = myPublications.GetItem("Bolt-1")

# Sets the meshed bolt support
myLinkAccess = myMeshedBolt.GetItem("SimLinkAccess")
myLinkAccess.AddLink ("MainSupport", mySupport)
...

```

Important : Note that the "MainSupport" string is a default value. For connection features you will have to use "PRD1", "PRD2", etc... to set the different supports and "Handler0D" or "Handler1D" for fasteners placements.

5. Sets the shell section supports from product structure

In this step UC sets the shell section supports from the product structure. It first retrieves usefull data to compose the link to the support and then adds this created link to the feature.

```
...
# Retrieves the ws3_bolt_part.2 part occurrence
myRootOccurrence = MyProductService.RootOccurrence

iMaxCnt = myRootOccurrenceOccurrences.Count

for index in range(1, iMaxCnt):
    if myRootOccurrenceOccurrences.Item(index).Name == "ws3_bolt_part.2":
        myPartOccurrence = myRootOccurrenceOccurrences.Item(index)

# Retrieves the ws3_bolt_part.2 part rep instance
myPartInstance = myPartOccurrence.InstanceOccurrenceOf
myPartInstanceRef = myPartInstance.ReferenceInstanceOf
myPartRepInstance = myPartInstanceRef.RepInstances.Item(1)

# Retrieves the ws3_bolt_part.2 part body
myPartRepInstanceRef = myPartRepInstance.ReferenceInstanceOf
myPart = myPartRepInstanceRef.GetItem("Part")
myPartBody = myPart.MainBody

# Composes link to the ws3_bolt_part.2 part body
myReference = myPart.CreateReferenceFromObject(myPartBody)
myLinkToFirstSupport = myProdService.ComposeLink(myPartOccurrence, myPartRepInstance, myReference)

# Sets the shell section first support
myLinkAccess = myShellSection.GetItem("SimLinkAccess")
myLinkAccess.AddLink ("MainSupport", myLinkToFirstSupport)

# Retrieves the ws3_bolt_part.3 part occurrence
for index in range(1, iMaxCnt):
    if myRootOccurrenceOccurrences.Item(index).Name == "ws3_bolt_part.3":
        myPartOccurrence = myRootOccurrenceOccurrences.Item(index)

# Retrieves the ws3_bolt_part.3 part rep instance
myPartInstance = myPartOccurrence.InstanceOccurrenceOf
myPartInstanceRef = myPartInstance.ReferenceInstanceOf
myPartRepInstance = myPartInstanceRef.RepInstances.Item(1)

# Retrieves the ws3_bolt_part.3 part body
myPartRepInstanceRef = myPartRepInstance.ReferenceInstanceOf
myPart = myPartRepInstanceRef.GetItem("Part")
myPartBody = myPart.MainBody

# Composes link to the ws3_bolt_part.3 part body
myReference = myPart.CreateReferenceFromObject(myPartBody)
myLinkToSecondSupport = myProdService.ComposeLink(myPartOccurrence, myPartRepInstance, myReference)

# Sets the shell section second support
myLinkAccess.AddLink ("MainSupport", myLinkToSecondSupport)
...
```

6. Replaces the first shell section support

In this step UC replaces the first shell section support.

```
...
# Retrieves the publication
myNewSupport = myPublications.GetItem("Pipe")

# Removes first support
myLinkAccess.RemoveLink ("MainSupport", myLinkToFirstSupport)

# Sets the new support
myLinkAccess.AddLink ("MainSupport", myNewSupport)
...
```

7. Removes all supports of the shell section

In this step UC removes all supports of the shell section.

```
...
myLinkAccess.RemoveAllLinks ("MainSupport")
...
```

Using ComposeLink method

This article explains how to use the ComposeLink method of the PLMProductService object. It will show the different way to create a link memory object in Python.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Launch the structural scenario App and create a structural simulation.
- Make sure that you have all of the licenses required for launching execution and perform a solve operation in the interactive session for a similar simulation to confirm the licenses.

Where to find the macro: [CAAScdfeaComposeLinkWithPublicationInVBSource.htm](#)

Where to find the macro: [CAAScdfeaComposeLinkWithoutPublicationInVBSource.htm](#)

This use case can be divided into seven steps:

1. [Retrieves the active Product](#)
2. [Creates link](#)

3. [Associates representation to FEM](#)
4. [Creates a Shell section](#)
5. [Sets the shell section attributes](#)

1. Retrieves the active Product

As a first step, the UC retrieves the active product.

Using Publication Without Publication

```
...
' As a first step, retrieve the product editor which is the activated one
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor

' Then retrieve the product service
Dim myProdService As PLMProductService
Set myProdService = myEditor.GetService("PLMProductService")

' Retrieve the Root Occurrence
Dim myRootOcc As VPMRootOccurrence
Set myRootOcc = myProdService.RootOccurrence

...
...

' As a first step, retrieve the product editor which is the activated one
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor

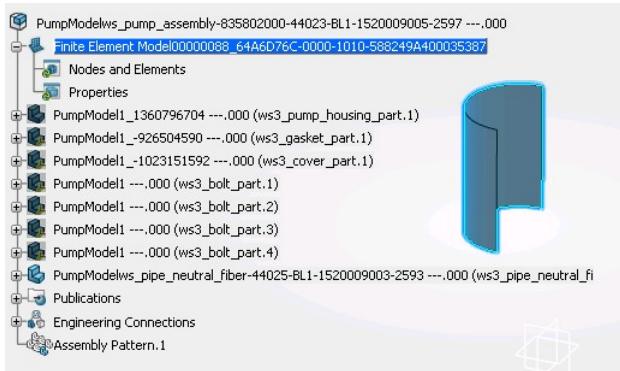
' Then retrieve the product service
Dim myProdService As PLMProductService
Set myProdService = myEditor.GetService("PLMProductService")

' Retrieve the Root Occurrence
Dim myRootOcc As VPMRootOccurrence
Set myRootOcc = myProdService.RootOccurrence
...
```

2. Creates link

In this step, the UC retrieves useful data to compose link to the support.

The created link, a 3D Shape in this case, is used to associate the Pipe part to the FemRep.



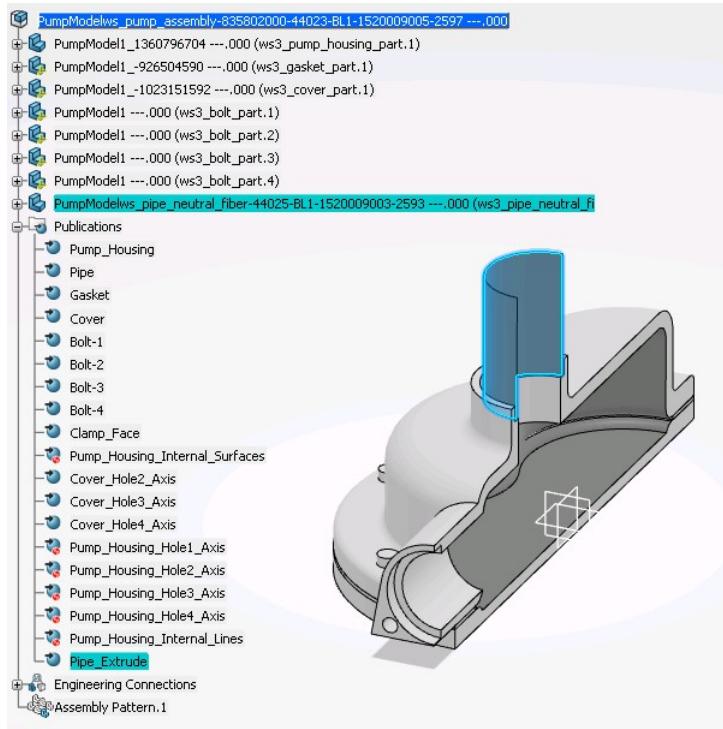
Using Publication Without Publication

```
...
' retrieve the publications set
Dim myPublications As VPMPublications
Set myPublications = myRootOcc.ReferenceRootOccurrenceOf.Publications

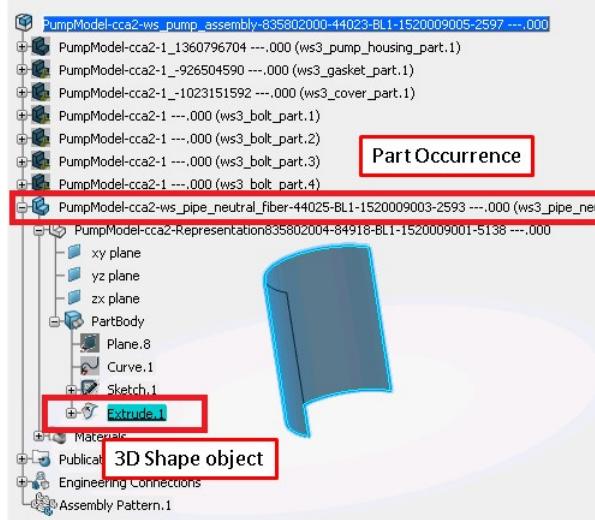
' retrieve the Pipe_Extrude publication, it is a 3D Shape
Dim myPublication As VPMPublication
Set myPublication = myPublications.GetItem("Pipe_Extrude")

' Create the link with ComposeLink method.
' As we use publication, we do not need to give the representation instance as the publication is under the product reference.
' Set as third argument the retrieved publication defining the feature needed to be linked
Dim myLink As AnyObject
Set myLink = myProdService.ComposeLink(myRootOcc, Nothing, myPublication)
...
```

Used publication is aggregated to the part wanted to be associated and the publication points a 3D Shape element. So it has the exact path of the needed 3D Shape.



```
...
' To create the correct link, retrieve the part occurrence, needed for the first argument
' Retrieve the part rep instance, needed for the second argument
' Creates a reference for the 3D Shape object needed for the third argument
' Be aware to retrieve the part occurrence and the part rep instance to whom the 3D Shape is associated
```



```
' Retrieve the root occurrence
Dim myRootOccurrence As VPMRootOccurrence
Set myRootOccurrence = myProdService.RootOccurrence

' Retrieve the ws3_pipe_neutral_fiber_part.1 part occurrence
Dim myPartOccurrence As PLMOccurrence

For i = 1 to myRootOccurrenceOccurrences.Count
    If myRootOccurrenceOccurrences.Item(i).Name = "ws3_pipe_neutral_fiber_part.1" Then
        Set myPartOccurrence = myRootOccurrenceOccurrences.Item(i)
    End if
Next

' Retrieve the ws3_pipe_neutral_fiber_part.1 part rep instance
Dim myPartInstance As VPMInstance
Set myPartInstance = myPartOccurrence.InstanceOccurrenceOf

Dim myPartInstanceRef As VPMReference
Set myPartInstanceRef = myPartInstance.ReferenceInstanceOf

Dim myPartRepInstance As VPMRepInstance
Set myPartRepInstance = myPartInstanceRef.RepInstances.Item(1)

' Retrieve the ws3_pipe_neutral_fiber_part.1 part body
Dim myPartRepInstanceRef As VPMRepReference
Set myPartRepInstanceRef = myPartRepInstance.ReferenceInstanceOf
```

```

Dim myPart As Part
Set myPart = myPartRepInstanceRef.GetItem("Part")

Dim myPartBody As Body
Set myPartBody = myPart.MainBody

' Retrieve the 3D shape Extrude.1
Dim myExtrude As AnyObject
Set myExtrude = myPart.FindObjectByName("Extrude.1")

Dim myReference As Reference
Set myReference = myPart.CreateReferenceFromObject(myExtrude)

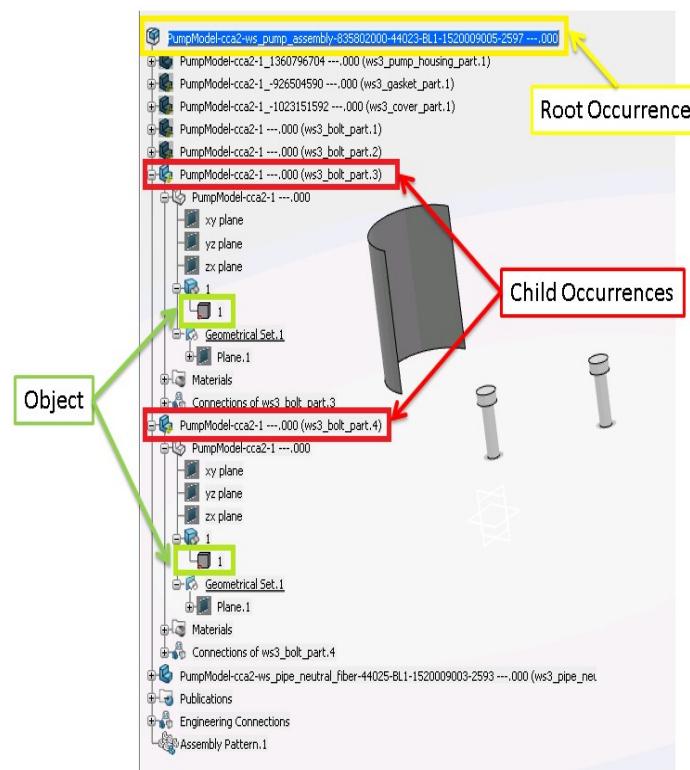
' Create the link with ComposeLink method.
Dim myLink As AnyObject
Set myLink = myProdService.ComposeLink(myPartOccurrence, myPartRepInstance, myReference)
...

```

Important :

To retrieve the needed path through the compose link method, correct arguments have to be given.

As first argument, it is necessary to give the part occurrence because sometimes the root occurrence could have two same part as child occurrence. Due to this, created myReference argument is difficult to locate correctly, even if the correct PartRepInstance is given, the reference will be the same for each same object. See an example on the image below.

**3. Associates representation to FEM**

In this step, UC creates the FEM rep and associates the representation to it

Using Publication Without Publication

```

...
' Retrieve the Product representation factory to create a FEM
Dim MyPrdRepFactory As SimPrdRepFactory
Set MyPrdRepFactory = myAssembly.GetItem("SimPrdRepFactory")

Dim MyFemRepRef As VPMRepReference
Set MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")

Dim MyFemRepRoot As SimFemRoot
Set MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")

' Add link to the FEM rep
MyFemRepRoot.AddAssociatedRep (myLink)
...

...
' Retrieve the Product representation factory to create a FEM
Dim MyPrdRepFactory As SimPrdRepFactory
Set MyPrdRepFactory = myAssembly.GetItem("SimPrdRepFactory")

Dim MyFemRepRef As VPMRepReference
Set MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")

Dim MyFemRepRoot As SimFemRoot
Set MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")

```

```
' Add link to the FEM rep
MyFemRepRoot.AddAssociatedRep (myLink)

...

```

Important : When using publication, we do not need to create a link in memory. The publication can directly be setted as argument
 MyFemRepRoot.AddAssociatedRep (myPublication)

4. Creates a Shell section

In this step, UC creates shell section

Using Publication Without Publication

```
...
' Retrieve the properties set
Dim myPropertiesSet As SimProperties
Set myPropertiesSet = MyFemRepRoot.GetSet("SimProperties")

' Create individual properties
Dim myShellSection As SimShellSection
Set myShellSection = myPropertiesSet.Add("SimShellSection")
...

...
' Retrieve the properties set
Dim myPropertiesSet As SimProperties
Set myPropertiesSet = MyFemRepRoot.GetSet("SimProperties")

' Create individual properties
Dim myShellSection As SimShellSection
Set myShellSection = myPropertiesSet.Add("SimShellSection")
...
```

5. Set the shell section attributes

In this step, UC sets attributes to the created shell section

Using Publication Without Publication

```
...
' Set the section support
Dim myLinkAccess As SimLinkAccess
Set myLinkAccess = myShellSection.GetItem("SimLinkAccess")

' Add the link created before
myLinkAccess.AddLink "MainSupport", myLink
myShellSection.UniformThickness = 1.0
...

...
' Set the section support
Dim myLinkAccess As SimLinkAccess
Set myLinkAccess = myShellSection.GetItem("SimLinkAccess")

' Add the link created before
myLinkAccess.AddLink "MainSupport", myLink
myShellSection.UniformThickness = 1.0
...
```

Important : When using publication, we do not need to create a link in memory. The publication can directly be setted as argument

myLinkAccess.AddLink ("MainSupport", myPublication)

Using ComposeLink method

This article explains how to use the ComposeLink method of the PLMProductService object. It will show the different way to create a link memory object in Python.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Launch the structural scenario App and create a structural simulation.
- Make sure that you have all of the licenses required for launching execution and perform a solve operation in the interactive session for a similar simulation to confirm the licenses.

Where to find the macro: [CAAScdfeaComposeLinkWithPublicationInPythonSource.htm](#)

Where to find the macro: [CAAScdfeaComposeLinkWithoutPublicationInPythonSource.htm](#)

This use case can be divided into seven steps:

1. [Retrieves the simulation and its product](#)
2. [Creates link](#)
3. [Associates representation to FEM](#)
4. [Creates a Shell section](#)
5. [Sets the shell section attributes](#)

1. Retrieves the simulation and its product

As a first step, the UC retrieves the active editor and retrieves the simulation model.

Using Publication Without Publication

```

...
# As a first step, retrieve the simulation editor which is the activated one
myEditor = CATIA.ActiveEditor

# Then retrieve the product service
myProdService = myEditor.GetService("PLMProductService")

# Retrieve the Root Occurrence
myRootOccurrence = myProdService.RootOccurrence
...

...
# As a first step, retrieve the simulation editor which is the activated one
myEditor = CATIA.ActiveEditor

# Then retrieve the product service
myProdService = myEditor.GetService("PLMProductService")

# Retrieve the PLMEntities
myEntities = myProdService.EditedContent

# Now retrieve the PLMEntity
myEntity = myEntities.Item(1)

# Get the simulation Root
MySimulationRoot = myEntity

# Now retrieve the Model
myModel = MySimulationRoot.Model

# UC Opens now the model to retrieve the root occurrence.
# The root occurrence has to be retrieved to access to the part occurrences.

oOpenService = CATIA.GetSessionService("PLMOpenService")
oOpenService.PLMOpen (myModel)

myProductEditor = CATIA.ActiveEditor
MyProductService = myProductEditor.GetService("PLMProductService")

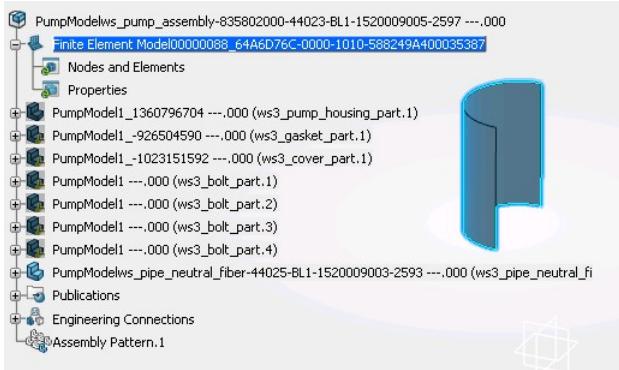
myRootOccurrence = MyProductService.RootOccurrence
...

```

2. Creates link

In this step, the UC retrieves usefull data to compose link to the support.

The created link, a 3D Shape in this case, is used to associate the Pipe part to the FemRep.



Using Publication Without Publication

```

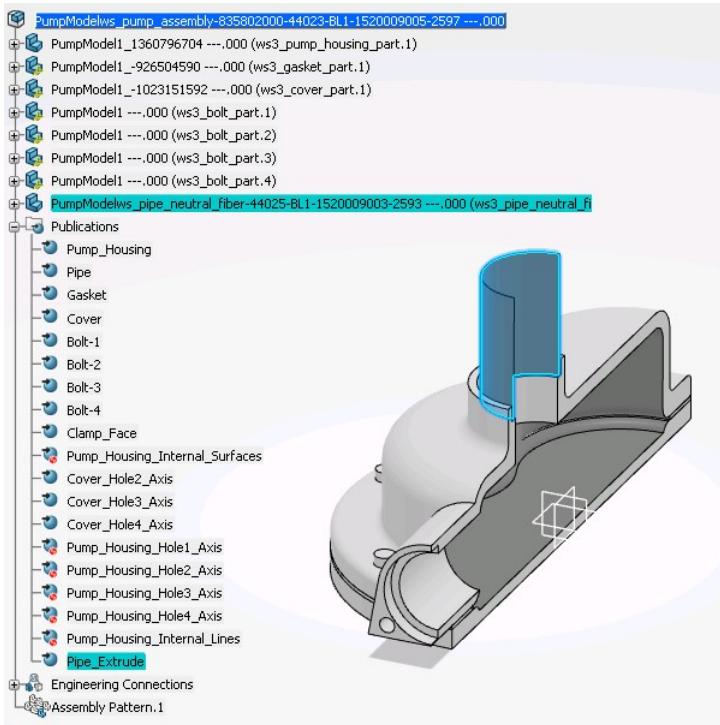
...
# retrieve the publications set
myPublications = myRootOccurrence.ReferenceRootOccurrenceOf.Publications

# retrieve the Pipe_Extrude publication, it is a 3D Shape
myPublication = myPublications.GetItem("Pipe_Extrude")

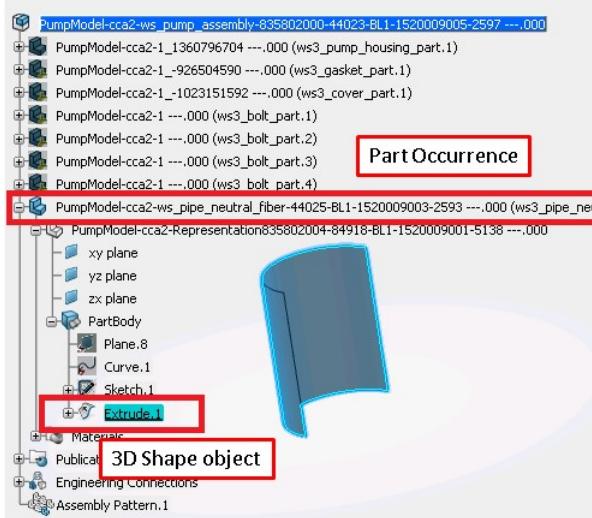
# Create the link with ComposeLink method.
# As we use publication, we do not need to give the representation instance as the publication is under the product reference.
# Set as third argument the retrieved publication defining the feature needed to be linked
myLink = myProdService.ComposeLink(myRootOccurrence, None, myPublication)
...

```

Used publication is aggregated to the part wanted to be associated and the publication points a 3D Shape element. So it has the exact path of the needed 3D Shape.



```
...
# To create the correct link, retrieve the part occurrence, needed for the first argument
# Retrieve the part rep instance, needed for the second argument
# Creates a reference for the 3D Shape object needed for the third argument
# Be aware to retrieve the part occurrence and the part rep instance to whom the 3D Shape is associated
```



```
# Retrieve the ws3_pipe_neutral_fiber_part.1 part occurrence
iMaxCnt = myRootOccurrenceOccurrences.Count

for index in range(1, iMaxCnt+1):
    if myRootOccurrenceOccurrences.Item(index).Name == "ws3_pipe_neutral_fiber_part.1":
        myPartOccurrence = myRootOccurrenceOccurrences.Item(index)

    # Retrieve the ws3_pipe_neutral_fiber_part.1 part rep instance
    myPartInstance = myPartOccurrence.InstanceOccurrenceOf
    myPartInstanceRef = myPartInstance.ReferenceInstanceOf
    myPartRepInstance = myPartInstanceRef.RepInstances.Item(1)

    # Retrieve the ws3_pipe_neutral_fiber_part.1 part body
    myPartRepInstanceRef = myPartRepInstance.ReferenceInstanceOf("Part")
    myPart = myPartRepInstanceRef.GetItem("Part")
    myPartBody = myPart.MainBody

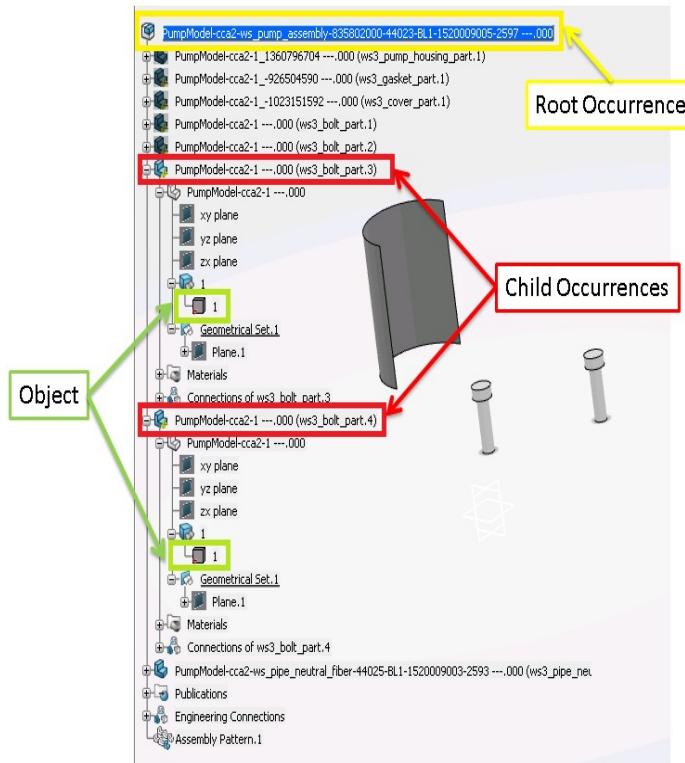
    # Retrieve the 3D shape Extrude.1
    myExtrude = myPart.FindObjectByName("Extrude.1")
    myReference = myPart.CreateReferenceFromObject(myExtrude)

    # Create the link with ComposeLink method.
    myLink = myProdService.ComposeLink(myPartOccurrence, myPartRepInstance, myReference)
    ...
```

Important :

To retrieve the needed path through the compose link method, correct arguments have to be given.

As First argument, it is necessary to give the part occurrence because sometimes the root occurrence could have two same part as child occurrence. Due to this, created myReference argument is difficult to locate correctly, even if the correct PartRepInstance is given, the reference will be the same for each same object. See an example on the image below.



3. Associates representation to FEM

In this step, UC creates the FEM rep and associates the representation to it

Using Publication Without Publication

```
...
# retrieve the Product representation factory to create a FEM
MyPrdRepFactory = myModel.GetItem("SimPrdRepFactory")
MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")
MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")

# Add link to the FEM rep
MyFemRepRoot.AddAssociatedRep (myLink)

...

...
# retrieve the Product representation factory to create a FEM
MyPrdRepFactory = myModel.GetItem("SimPrdRepFactory")
MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")
MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")

# Add link to the FEM rep
MyFemRepRoot.AddAssociatedRep (myLink)
...
```

Important : When using publication, we do not need to create a link in memory. The publication can directly be setted as argument

```
MyFemRepRoot.AddAssociatedRep (myPublication)
```

4. Creates a Shell section

In this step, UC creates shell section

Using Publication Without Publication

```
...
# Retrieve the properties set
myPropertiesSet = MyFemRepRoot.GetSet("SimProperties")

# Create individual properties
myShellSection = myPropertiesSet.Add("SimShellSection")
...

...
# Retrieve the properties set
myPropertiesSet = MyFemRepRoot.GetSet("SimProperties")

# Create individual properties
myShellSection = myPropertiesSet.Add("SimShellSection")
...
```

5. Set the shell section attributes

In this step, UC sets attributes to the created shell section

Using Publication Without Publication

```
...
# Set the section support
myLinkAccess = myShellSection.GetItem("SimLinkAccess")
# Add the link created before
myLinkAccess.AddLink ("MainSupport", myLink)
myShellSection.UniformThickness = 1.0
...

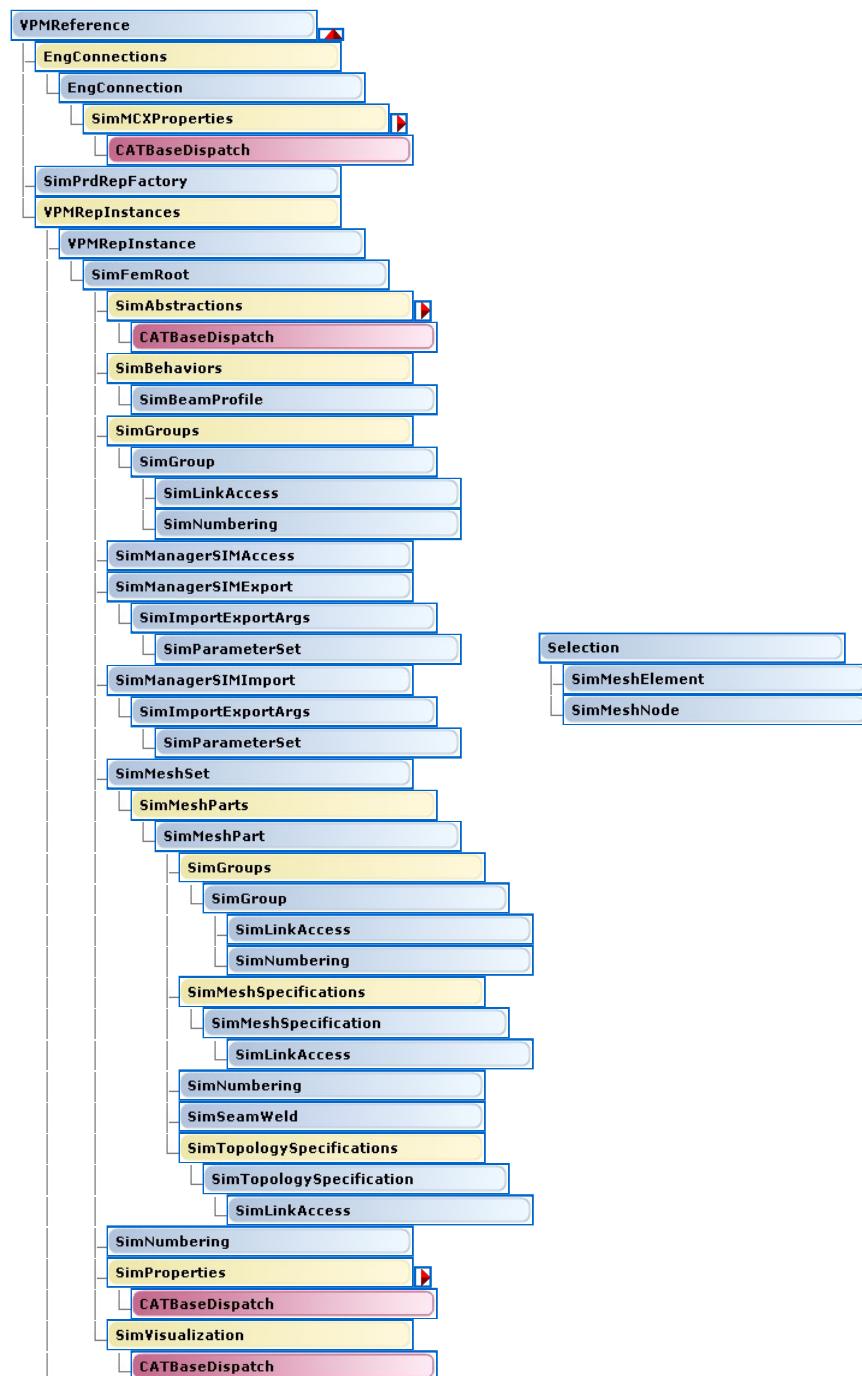
...
# Set the section support
myLinkAccess = myShellSection.GetItem("SimLinkAccess")
# Add the link created before.
myLinkAccess.AddLink ("MainSupport", myLink)
myShellSection.UniformThickness = 1.0
...
```

Important : When using publication, we do not need to create a link in memory. The publication can directly be setted as argument

```
myLinkAccess.AddLink ("MainSupport", myPublication)
```

Multiphysics Model Creation Object Model Map

See Also [Legend](#)





To retrieve the root **VPMReference** object, see [Browsing Product Contents](#).

Use the **GetItem** method to return the **SimLinkAccess** object from its aggregating object, such as the **SimGroup** object.

VBA Python

```

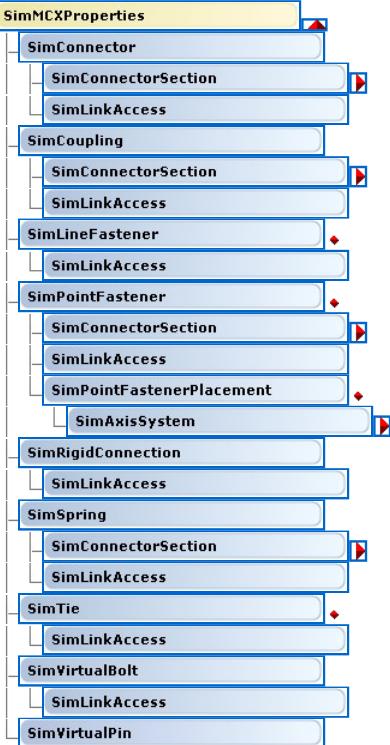
Dim oSimGroup As SimGroup
...
Dim oSimLinkAccess As SimLinkAccess
Set oSimLinkAccess = oSimGroup.GetItem("SimLinkAccess")

...
oSimLinkAccess = oSimGroup.GetItem("SimLinkAccess")

```

SimMCXProperties Collection

See Also [Legend](#)



♦ Available with the *RBM - Simulation Scripting* product only.

Use the **GetItem** method to return the **SimLinkAccess** object from its aggregating object, such as the **SimVirtualBolt** object.

VBA Python

```

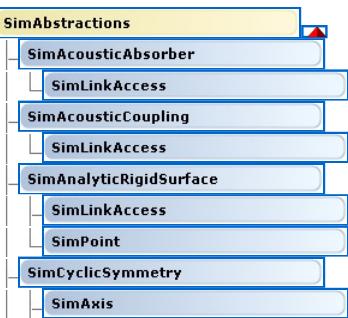
Dim oSimVirtualBolt As SimVirtualBolt
...
Dim oSimLinkAccess As SimLinkAccess
Set oSimLinkAccess = oSimVirtualBolt.GetItem("SimLinkAccess")

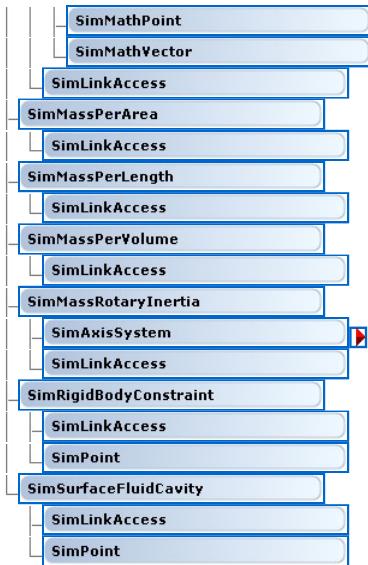
...
oSimLinkAccess = oSimVirtualBolt.GetItem("SimLinkAccess")

```

SimAbstractions Collection

See Also [Legend](#)





Use the **GetItem** method to return the **SimLinkAccess** object from its aggregating object, such as the **SimSurfaceFluidCavity** object.

VBA Python

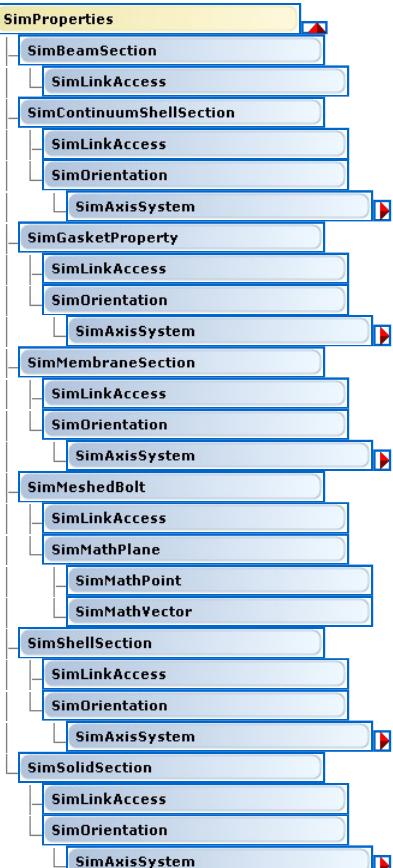
```

Dim oSimSurfaceFluidCavity As SimSurfaceFluidCavity
...
Dim oSimLinkAccess As SimLinkAccess
Set oSimLinkAccess = oSimSurfaceFluidCavity.GetItem("SimLinkAccess")

...
oSimLinkAccess = oSimSurfaceFluidCavity.GetItem("SimLinkAccess")
  
```

SimProperties Collection

See Also [Legend](#)



Use the **GetItem** method to return the **SimLinkAccess** object from its aggregating object, such as the **SimMeshedBolt** object.

VBA Python

```

Dim oSimMeshedBolt As SimMeshedBolt
  
```

```

...
Dim oSimLinkAccess As SimLinkAccess
Set oSimLinkAccess = oSimMeshedBolt.GetItem("SimLinkAccess")

...
oSimLinkAccess = oSimMeshedBolt.GetItem("SimLinkAccess")

```

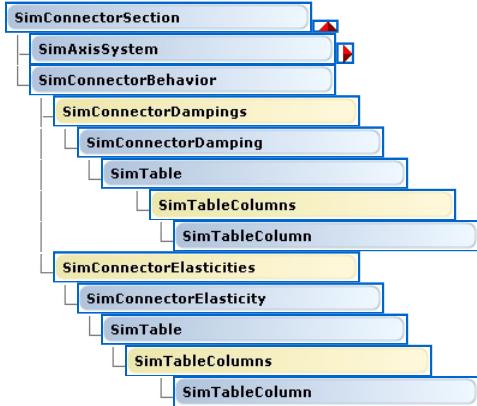
SimAxisSystem Object

See Also [Legend](#)



SimConnectorSection Object

See Also [Legend](#)



Creating a FEM rep and managing its associated shapes

This use case primarily focuses on the methodology to create a FEM representation and to manage associated shapes.

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdfeaProductWithoutFEM.3dxml` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaModeling\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: [CAAScdfeaCreateFEMSource.htm](#)

This use case can be divided in seven steps:

1. [Searches and opens model which is named as "CAAScdfeaProductWithoutFEM"](#)
2. [Retrieves the root product](#)
3. [Retrieves the PartBody](#)
4. [Creates the FEM representation and retrieve its root object](#)
5. [Associates the PartBody with the FEM representation](#)
6. [Checks if the FEM representation have at least one associated shape](#)
7. [Removes an associated shapes](#)

1. Searches and opens model which is named as "CAAScdfeaProductWithoutFEM"

As a first step, the UC retrieves the model named "CAAScdfeaProductWithoutFEM", loads it, and returns object of the Editor.

```

...
Dim myEditor As Editor
OpenProduct myEditor
...

```

The function `OpenProduct` returns `MyEditor`, an Editor object. After searching and opening of "CAAScdfeaProductWithoutFEM" product from underlying database the current active editor is returned in `MyEditor`.

2. Retrieves the root product

As a next step, the UC retrieves the root product object from the returned editor.

```

...
Dim MyRootOcc As VPMRootOccurrence
Set MyRootOcc = MyEditor.ActiveObject

Dim MyRootProduct As VPMReference
Set MyRootProduct = MyRootOcc.PLMEentity
...

```

3. Retrieves the PartBody

In this step UC retrieves the main body of the 3D shape aggregated by the root product.

```
...
Dim MyContext As PLMProductService
Set MyContext = MyEditor.GetService("PLMProductService")

Dim MyPart As Part
Dim MyPartInstance As VPMRepInstance
Dim MyBody As Body
For Each MyEntity In MyContext.EditedContent
    Dim MyRef As VPMReference
    Set MyRef = MyEntity
    Dim MyReps As VPMRepInstances
    Set MyReps = MyRef.RepInstances

    For Each MyRep In MyReps
        Dim MyRepRef As VPMRepReference
        Set MyRepRef = MyRep.ReferenceInstanceOf
        Dim attr As String
        attr = MyRepRef.GatAttributValue("V_discipline")

        If (attr = "Design") Then
            Set MyPart = MyRepRef.GetItem("Part")
            Set MyPartInstance = MyRep
            Set MyBody = MyPart.MainBody
        End If
    Next
...

```

4. Creates the FEM representation and retrieve its root object

In this step UC creates a new FEM representation object. To perform the creation, a simulation representation factory has to be retrieved on the reference object which will aggregate the FEM representation.

```
...
Dim MyPrdRepFactory As SimPrdRepFactory
Set MyPrdRepFactory = MyRootProduct.GetItem("SimPrdRepFactory")
Dim MyFemRepRef As VPMRepReference
Set MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")
Dim MyFemRepRoot As SimFemRoot
Set MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")
...

```

5. Associates the PartBody with the FEM representation

In this step UC appends the main body previously retrieved to the associated shapes of the FEM representation.

```
...
Dim MyProductService As PLMProductService
Set MyProductService = MyEditor.GetService("PLMProductService")

Dim MyReference As Reference
Set MyReference = MyPart.CreateReferenceFromObject(MyBody)
Dim MyLink As AnyObject
Set MyLink = MyProductService.ComposeLink(MyRootOcc, MyPartInstance, MyReference)
MyFemRepRoot.AddAssociatedRep MyLink
...

```

6. Checks if the FEM representation have at least one associated shape

In this step UC checks if the FEM representation has at least one associated shape.

```
...
Dim bHasAnAssociatedRep As Boolean
bHasAnAssociatedRep = MyFemRepRoot.HasAnAssociatedRep
If (bHasAnAssociatedRep = False) Then
    MsgBox "Error while associating the PartBody to the FEM Rep!"
    Exit Sub
End If
...

```

7. Removes an associated shapes

In this step UC removes the main body previously retrieved from the associated shapes of the new FEM representation.

```
...
MyFemRepRoot.RemoveAssociatedRep MyLink
bHasAnAssociatedRep = MyFemRepRoot.HasAnAssociatedRep
If (bHasAnAssociatedRep = True) Then
    MsgBox "Error while removing the associated shape from the FEM Rep!"
    Exit Sub
End If
...

```

Creating a FEM rep and managing its associated shapes

This use case primarily focuses on the methodology to create a FEM representation and to manage associated shapes.

Before you begin: Note that:

- You should first launch CATIA and import the CAAScdfeaProductWithoutFEM.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaModeling\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- You should open the product called CAAScdfeaProductWithoutFEM.
- Launch the structural scenario App and create a structural simulation.

Where to find the macro: [CAAScdfeaCreateFEMWithPythonSource.htm](#)

This use case can be divided in seven steps:

1. [Retrieves the simulation and its product](#)
2. [Retrieves the PartBody](#)
3. [Creates the FEM representation and retrieves its root object](#)
4. [Associates the PartBody with the FEM representation](#)
5. [Checks if the FEM representation have at least one associated shape](#)
6. [Removes an associated shapes](#)
1. **Retrieve the simulation and its product**

As a first step, the UC retrieves the simulation and its product from the simulation editor

```
...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...
```

2. Retrieves the PartBody

In this step UC retrieves the main body of the 3D shape aggregated by the simulation's Model.

```
...
myModelRepInstances = myModel.RepInstances

for myPartRepInstance in myModelRepInstances:
    myPartRepInstanceRef = myPartRepInstance.ReferenceInstanceOf
    attr = myPartRepInstanceRef.GetAttributeValue("V_discipline")
    if attr == "Design":
        myPart = myPartRepInstanceRef.GetItem("Part")
        myPartRepInstance = myPartRepInstance
        myBody = myPart.MainBody
...

```

3. Creates the FEM representation and retrieve its root object

In this step UC creates a new FEM representation object.

To perform the creation, a simulation representation factory has to be retrieved on the reference object which will aggregate the FEM representation.

```
...
MyPrdRepFactory = myModel.GetItem("SimPrdRepFactory")
MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")
MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")
...
```

4. Associates the PartBody with the FEM representation

In this step UC appends the main body previously retrieved to the associated shapes of the FEM representation.

First, the UC opens the model and retrieves the product editor and the root occurrence

```
...
oOpenService = CATIA.GetService("PLMOpenService")
oOpenService.PLMOpen (myModel)
myProductEditor = CATIA.ActiveEditor
myRootOcc = myProductEditor.ActiveObject
```

Now it creates the link to associate with the FEM

```
MyReference = myPart.CreateReferenceFromObject(myBody)

MyLink = myProdService.ComposeLink(myRootOcc, myPartRepInstance, MyReference)
MyFemRepRoot.AddAssociatedRep (MyLink)
...

```

5. Checks if the FEM representation have at least one associated shape

In this step UC checks if the FEM representation has at least one associated shape.

```
...
bHasAnAssociatedRep = MyFemRepRoot.HasAnAssociatedRep
if bHasAnAssociatedRep == False:
    print "Error while associating the part body to the fem rep !"
...

```

6. Removes an associated shapes

In this step UC removes the main body previously retrieved from the associated shapes of the new FEM representation.

```
...
bHasAnAssociatedRep = MyFemRepRoot.HasAnAssociatedRep
if bHasAnAssociatedRep == True:
    print "Error while removing the associated shape from the FEM Rep!"
...

```

Creating an Assembly of FEMs

This use case primarily focuses on the methodology to create an assembly of FEMs.

Before you begin: Note that:

- You should first launch CATIA and import the CAAScdfeaFEMAssembly.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaModeling\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Then, you should open the "CAAScdfeaFEMAssembly" physical product.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaFEMAssemblySource.htm](#)

This use case can be divided in six steps:

1. [Retrieves the active editor](#)
2. [Retrieves the root product](#)
3. [Creates a FEM Rep and retrieve its root object](#)
4. [Creates a link between the created FEM and the first occurrence FEM](#)
5. [Creates a link between the created FEM and the second occurrence FEM](#)
6. [Updates the assembly FEM Rep](#)

1. Retrieves the active editor

As a first step, the UC retrieves the active editor.

```
...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor
...
```

2. Retrieves the root product

In the next step, the UC retrieves the root product from the active editor.

```
...
Dim MyRootOcc As VPMRootOccurrence
Set MyRootOcc = MyEditor.ActiveObject

Dim MyRootProduct As VPMReference
Set MyRootProduct = MyRootOcc.PLMEEntity
...
```

3. Creates a FEM Rep and retrieve its root object

In the next step the UC creates the assembly FEM Rep which contains links to the other FEM Rep. The assembly FEM Rep is aggregated under the root product.

```
...
Dim MyPrdRepFactory As SimPrdRepFactory
Set MyPrdRepFactory = MyRootProduct.GetItem("SimPrdRepFactory")
Dim MyFemRepRef As VPMRepReference
Set MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")
Dim MyFemRepRoot As SimFemRoot
Set MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")
...
```

4. Creates a link between the created FEM and the first occurrence FEM

In this step the UC retrieves the FEM Rep which is aggregated by the first product occurrence. Then the UC creates a path element that will be used to create the link between the retrieved FEM Rep and the assembly FEM Rep.

```
...
'Retrieve the first product occurrence
Dim myRootOccurrences As VPMOccurrences
Set myRootOccurrences = MyRootOcc.Occurrences
Dim myOccurrence As VPMOccurrence
Set myOccurrence = myRootOccurrences.Item(1)

'Retrieve the occurrence representation instance
Dim MyRepInstances As VPMRepInstances
Set MyRepInstances = myOccurrence.InstanceOccurrenceOf.ReferenceInstanceOf.RepInstances

'Retrieve the FEM Rep
Dim attr As String
Dim myRepRef As VPMRepReference
Dim MyLink As AnyObject
For Each myRep In MyRepInstances
    Set myRepRef = myRep.ReferenceInstanceOf
    attr = myRepRef.GetAttributeValue("V_discipline")
    If (attr = "FEM") Then
        'Create a path object from the root occurrence to the FEM Rep
        Set MyLink = MyProductService.ComposeLink(myOccurrence, myRep, Nothing)
    End If
Next

'Create the link between the assembly FEM Rep and the retrieved FEM Rep
MyFemRepRoot.AddChild MyLink
...
```

5. Creates a link between the created FEM and the second occurrence FEM

In a manner similar to the one shown above, the UC retrieves the FEM Rep which is aggregated by the second product and creates the path element from the root occurrence to the retrieved FEM Rep.

```
...
'Retrieve the FEM Rep of the second product occurrence
Set myOccurrence = myRootOccurrences.Item(2)
Set MyRepInstances = myOccurrence.InstanceOccurrenceOf.ReferenceInstanceOf.RepInstances
For Each myRep In MyRepInstances
    Set myRepRef = myRep.ReferenceInstanceOf
    attr = myRepRef.GetAttributeValue("V_discipline")
    If (attr = "FEM") Then
        'Create a path object from the root occurrence to the FEM Rep
        Set MyLink = MyProductService.ComposeLink(myOccurrence, myRep, Nothing)
    End If

```

```
Next
```

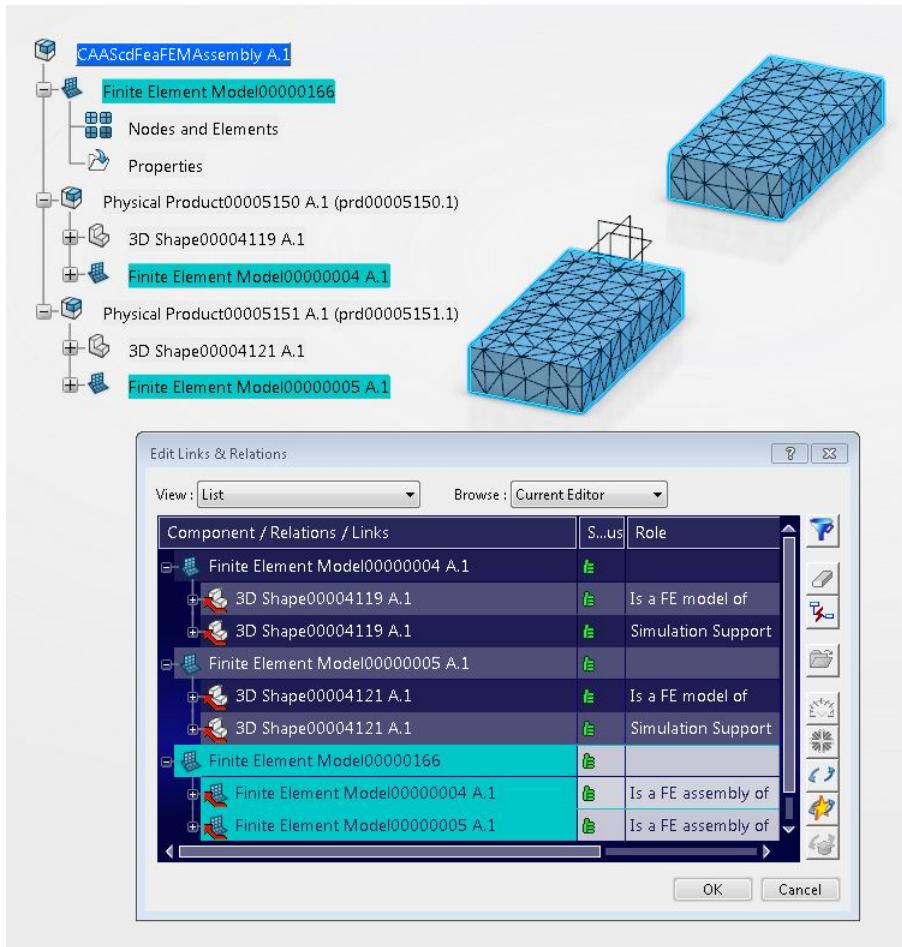
```
'Create the link between the assembly FEM Rep and the retrieved FEM Rep
MyFemRepRoot.AddChild MyLink
...
```

6. Updates the assembly FEM Rep

In this step the UC updates the assembly FEM Rep.

```
...
MyFemRepRoot.Update
...
```

Fig.1: Generated FEM Rep



Creating an Assembly of FEMs

This use case primarily focuses on the methodology to create an assembly of FEMs.

Before you begin: Note that:

- You should first launch CATIA and import the CAAScdFeaFEMAssembly.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdFeaModeling\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Then, you should open the "CAAScdFeaFEMAssembly" physical product as advanced with all representations.
- Create a structural simulation with launching the structural scenario apps.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdFeaFEMAssemblyWithPythonSource.htm](#)

This use case can be divided in six steps:

1. [Retrieves the simulation and its product](#)
2. [Gets the root occurrence](#)
3. [Creates a FEM Rep and retrieve its root object](#)
4. [Creates a link between the created FEM and the first occurrence FEM](#)
5. [Creates a link between the created FEM and the second occurrence FEM](#)
6. [Updates the assembly FEM Rep](#)

1. Retrieves the simulation and its product

As a first step, the UC retrieves the simulation and its product from the simulation editor.

```
...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMPProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
```

```

MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...

```

2. Gets the root occurrence

In the next step, the UC retrieves the root occurrence and the root product from the product editor and gets the PLMProductService associated with this editor.

```

...
oOpenService = CATIA.GetSessionService("PLMOpenService")
oOpenService.PLMOpen (myModel)
myProductEditor = CATIA.ActiveEditor
myRootOcc = myProductEditor.ActiveObject
myRootProduct = myRootOcc.PLMEentity
myProductService = myProductEditor.GetService("PLMProductService")
...

```

3. Creates the FEM representation and retrieves its root object.

In the next step the UC creates the assembly FEM Rep which contains links to the other FEM Rep. The assembly FEM Rep is aggregated under the root product.

```

...
MyPrdRepFactory = myRootProduct.GetItem("SimPrdRepFactory")
MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")
MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")
...

```

4. Creates a link between the created FEM and the first occurrence FEM

In this step the UC retrieves the FEM Rep which is aggregated by the first product occurrence. Then the UC creates a path element that will be used to create the link between the retrieved FEM Rep and the assembly FEM Rep.

```

...
# Retrieves the first product occurrence
myOccurrence = myRootOccurrences.Item(1)

# Retrieve the occurrence representation instance
MyRepInstances = myOccurrence.InstanceOccurrenceOf.ReferenceInstanceOf.RepInstances

# Retrieves the FEM Rep
for myRep in MyRepInstances :
    myRepRef = myRep.ReferenceInstanceOf
    attr = myRepRef.GetAttributeValue("V_discipline")
    if attr == "FEM":
        # Create a path object from the root occurrence to the FEM Rep
        MyLink = myProductService.ComposeLink(myOccurrence, myRep, None)

# Create the link between the assembly FEM Rep and the retrieved FEM Rep
MyFemRepRoot.AddChild (MyLink)
...

```

5. Creates a link between the created FEM and the second occurrence FEM

In a manner similar to the one shown above, the UC retrieves the FEM Rep which is aggregated by the second product and creates the path element from the root occurrence to the retrieved FEM Rep.

```

...
# Retrieves the second product occurrence
myOccurrence2 = myRootOccurrences.Item(2)

# Retrieve the occurrence representation instance
MyRepInstances2 = myOccurrence2.InstanceOccurrenceOf.ReferenceInstanceOf.RepInstances

# Retrieves the FEM Rep
for myRep2 in MyRepInstances2 :
    myRepRef2 = myRep2.ReferenceInstanceOf
    attr = myRepRef2.GetAttributeValue("V_discipline")
    if attr == "FEM":
        # Create a path object from the root occurrence to the FEM Rep
        MyLink2 = myProductService.ComposeLink(myOccurrence2, myRep2, None)

# Create the link between the assembly FEM Rep and the retrieved FEM Rep
MyFemRepRoot.AddChild (MyLink2)
...

```

6. Updates the assembly FEM Rep

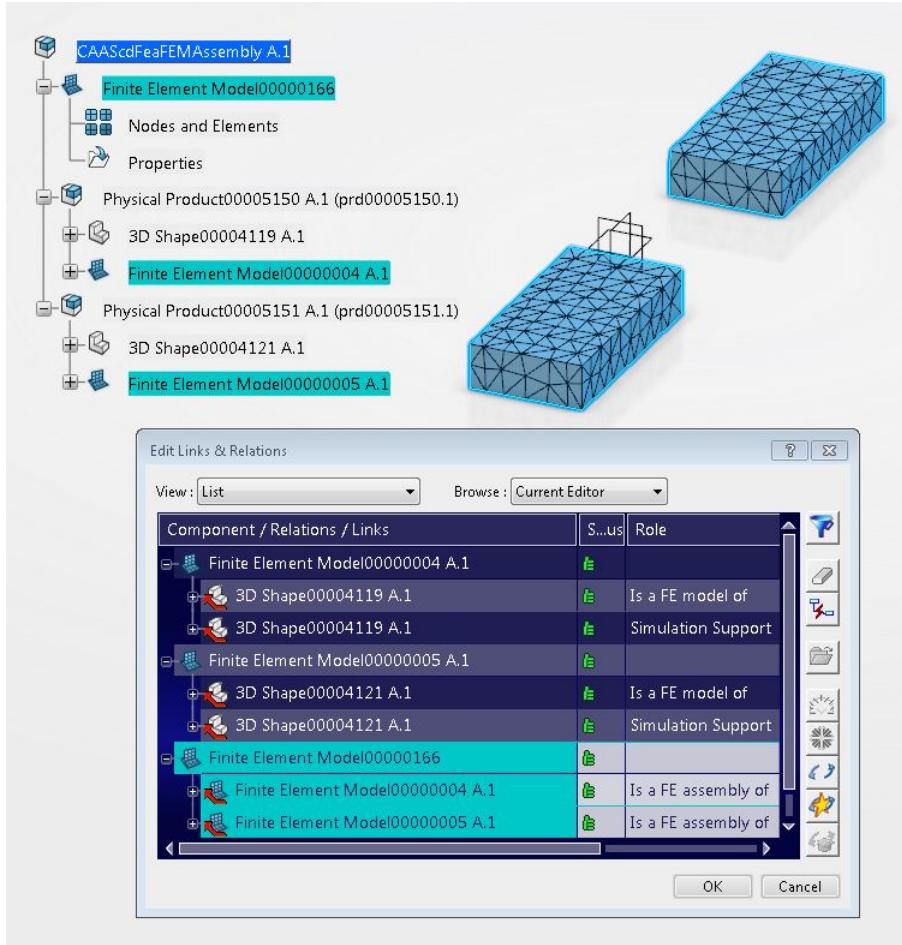
In this step the UC updates the assembly FEM Rep.

```

...
MyFemRepRoot.Update
...

```

Fig.1: Generated FEM Rep



Mesh Parts and Their Attributes

Technical Article

Abstract

The following reference article describes the different types of mesh parts, their attributes and usage.

This paper explains two types of mesh parts:

- The Beam Mesher. See [Beam Mesh Part Attributes](#). *The description of all attributes of the Beam mesh part.*
- The Coating 1D Mesher. See [Coating 1D Mesh Part Attributes](#). *The description of all attributes of the Coating 1D mesh part.*
- The Coating 2D Mesher. See [Coating 2D Mesh Part Attributes](#). *The description of all attributes of the Coating 2D mesh part.*
- The Extrusion Offset. See [Extrusion Offset Mesh Part Attributes](#). *The description of all attributes of the Extrusion Offset mesh part.*
- The Extrusion Rotation. See [Extrusion Rotation Mesh Part Attributes](#). *The description of all attributes of the Extrusion Rotation mesh part.*
- The Extrusion Spine. See [Extrusion Spine Mesh Part Attributes](#). *The description of all attributes of the Extrusion Spine mesh part.*
- The Extrusion Symmetry. See [Extrusion Symmetry Mesh Part Attributes](#). *The description of all attributes of the Extrusion Symmetry mesh part.*
- The Extrusion Translation. See [Extrusion Translation Mesh Part Attributes](#). *The description of all attributes of the Extrusion Translation mesh part.*
- The Hex-Dominant Mesher. See [Hex-Dominant Mesher Attributes](#). *The description of all attributes of the Hex-Dominant mesh part.*
- The Octree Mesher. See [Octree Mesh Part Attributes](#). *The description of all attributes of the Octree mesh part.*
- The Offset Mesher. See [Offset Mesh Part Attributes](#). *The description of all attributes of the Offset mesh part.*
- The Partition Hex Mesher. See [Partition Hex Mesh Part Attributes](#). *The description of all attributes of the Partition Hex mesh part.*
- The Quadrangle Surface Mesher. See [Surface Quadrangle Mesh Part Attributes](#). *The description of all attributes of the Surface Quadrangle mesh part.*
- The Rotation Mesher. See [Rotation Mesh Part Attributes](#). *The description of all attributes of the Rotation mesh part.*
- The Sweep 3D Mesher. See [Sweep 3D Mesh Part Attributes](#). *The description of all attributes of the Sweep 3D mesh part.*
- The Symmetry Mesher. See [Symmetry Mesh Part Attributes](#). *The description of all attributes of the Symmetry mesh part.*
- The Tetrahedron Mesher. See [Tetrahedron Mesh Part Attributes](#). *The description of all attributes of the Tetrahedron mesh part.*
- The Tetrahedron Filler Mesher. See [Tetrahedron Filler Mesh Part Attributes](#). *The description of all attributes of the Tetrahedron Filler mesh part.*
- The Translation Mesher. See [Translation Mesh Part Attributes](#). *The description of all attributes of the Translation mesh part.*
- The Triangle Surface Mesher. See [Surface Triangle Mesh Part Attributes](#). *The description of all attributes of the Surface Triangle mesh part.*

This paper describes also how to access to:

- [Mesh Part Creation](#)
- [Global Attributes](#)
- [Local Topology Specifications](#)
- [Local Mesh Specifications](#)
- [Meshing Rules](#)
- [Mesh Quality](#).

Mesh Part creation

Mesh Part creation is managed the `Add` method of the mesh part collection object.

A "late type" must be given to indicate which type of mesh part is to be created:

Mesh part	Late type
Beam	CATFmtBeamRulesMesher
Coating 2D	CATFmtCoating2DMesher
Extrusion Offset	CATFmtExtrusionOffsetMesher
Extrusion Rotation	CATFmtExtrusionRotationMesher
Extrusion Spine	CATFmtExtrusionSpineMesher
Extrusion Symmetry	CATFmtExtrusionSymmetryMesher
Extrusion Translation	CATFmtExtrusionTranslationMesher
Hex-Dominant Mesher	CATFmtHexMesher
Octree 2D	CATFmtOctree2DRulesMesher
Octree 3D	CATFmtOctree3DRulesMesher
Offset	CATFmtOffsetMesher
Partition Hex Mesher	CATFmtPartitionHexMesher
Quadrangle Surface	CATFmtSurfQuadMesher
Rotation	CATFmtRotationMesher
Sweep 3D	CATFmtSweep3DMesher
Symmetry	CATFmtSymmetryMesher
Tetrahedron	CATFmt3DRulesMesher
Tetrahedron Filler	CATFmtGHS3DMesher
Translation	CATFmtTranslationMesher
Triangle Surface	CATFmtSurfTriaMesher

Global Attributes

Global attributes is managed using `SimMeshPart` object:

- `SetAttributeValue (iSetType As String, iAttribute As String, iValue)` to set an attribute value.
- `GetAttributeValue (iSetType As String, iAttribute As String, oValue)` to get an attribute value.

`iAttribute` is the global attribute's name and `iSetType` is the type of specification:

- `iSetType = "Mesh"` for Global Mesh Attributes.
- `iSetType = "Topology"` for Global Topology Attributes.

Local Topology Specifications

Local topology specifications are created by the `Add` method of the `SimTopologySpecifications` object.

- `Add (iType As String)` with:
 - `iType` is the late type of the topology specification to create.

The `Item` and the `Remove` methods work with an index which is the position of the specification in the collection. The position index starts from 1.

The support of the local topology specification is managed by the `SimLinkAccess` object:

- `AddLink (iName As String, iLink As AnyObject)`
- `RemoveAllLinks (iName As String)`
- `RemoveLink (iName As String, iLink As AnyObject)`
 - `iName` is the name of the link container, here it's "`ConnectorList`"

The specification values are managed by the following methods:

- `GetAttributeValue (iAttribute As String, oValue)`
- `SetAttributeValue (iAttribute As String, iValue)`
 - `iAttribute` is the attribute name to set/retrieve.

Local Mesh Specifications

Local mesh specifications are created by the `Add` method of the `SimMeshSpecifications` object.

- `Add (iType As String)` with:
 - `iType` is the late type of the mesh specification to create.

The `Item` and the `Remove` methods work with an index which is the position of the specification in the collection. The position index starts from 1.

The support of the local topology specification is managed by the `SimLinkAccess` object:

- `AddLink (iName As String, iLink As AnyObject)`
- `RemoveAllLinks (iName As String)`
- `RemoveLink (iName As String, iLink As AnyObject)`
 - `iName` is the name of the link container, here it's "`ConnectorList`"

The specification values are managed by the following methods:

- `GetAttributeValue (iAttribute As String, oValue)`
- `SetAttributeValue (iAttribute As String, iValue)`
 - `iAttribute` is the attribute name to set/retrieve.

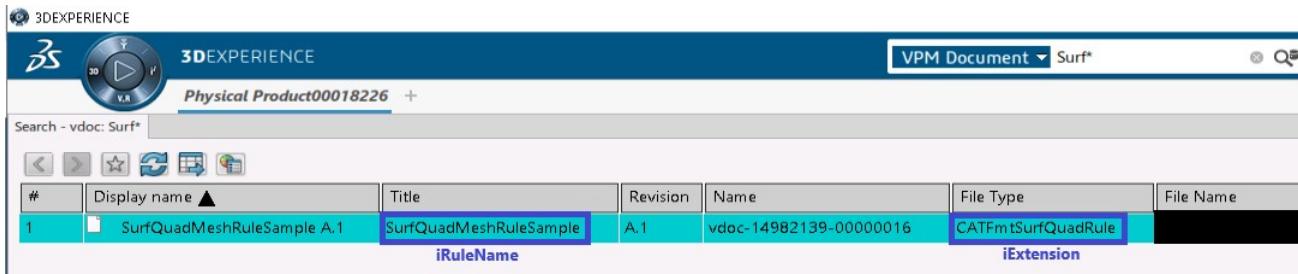
Meshing Rules

A Meshing Rule can be used to manage the mesh part parameters by using `SetRule` method of the `SimMeshPart` object.

- `SetRule (iRuleName As String, iExtension As String)` with:

- o **iRuleName** is the name of the meshing rule.
- o **iExtension** is the extension of the meshing rule.

Fig.1: Meshing Rule PLM Attributes



The screenshot shows the 3DEXPERIENCE VPM Document interface. At the top, there's a toolbar with icons for search, refresh, and other document operations. Below the toolbar, the title bar displays "Physical Product00018226". The main area is a table titled "Meshing Rule PLM Attributes". The table has columns: #, Display name, Title, Revision, Name, File Type, and File Name. A row is shown with the following values:

#	Display name	Title	Revision	Name	File Type	File Name
1	SurfQuadMeshRuleSample A.1	SurfQuadMeshRuleSample	A.1	vdoc-14982139-00000016	CATFm tSurfQuadRule	iExtension

Mesh Quality

The mesh quality analysis can be exported from the `SimMeshPart` or `SimFemRoot` objects.
 If the export is called on a `SimMeshPart`, the mesh quality analysis will be done on the mesh part only.
 If the export is called on the `SimFemRoot`, the mesh quality analysis will be done on the complete mesh set.

The mesh quality analysis can be exported by using the `CreateCSVFile` method of the `SimMeshQuality` object. This object can be retrieved by calling `GetItem` ("`SimMeshQuality`") method on the `SimMeshPart` or `SimFemRoot` objects.

- `CreateCSVFile (iPath As String, iOverwrite As Boolean)` with:
 - o **iPath** is the full path with the file name and its extension where to export the CSV file.
 - o **iOverwrite** to allow to overwrite an existing file.

History

Version: 1 [Oct 2014] Document created

Creating Mesh Part

This use case primarily focuses on the methodology to create a Mesh Part.

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdfeaProductWithoutFEM.3dxml` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaModeling\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaCreateMPSource.htm](#)

This use case can be divided in nine steps:

1. [Searches and opens model which is named as "CAAScdfeaProductWithoutFEM"](#)
2. [Retrieves the root product](#)
3. [Retrieves the PartBody](#)
4. [Creates the FEM representation and retrieve its root object](#)
5. [Retrieves the mesh set](#)
6. [Creates a Mesh Part](#)
7. [Creates a local topology specification](#)
8. [Creates a local mesh specification](#)
9. [Generates the mesh](#)

1. Searches and opens model which is named as "CAAScdfeaProductWithoutFEM"

As a first step, the UC retrieves a model "CAAScdfeaProductWithoutFEM" from DB and loads it and returns object of the Editor.

```
...
Dim myEditor As Editor
OpenProduct MyEditor
...
```

The function `OpenProduct` returns `MyEditor`, an Editor object. After searching and opening of "CAAScdfeaProductWithoutFEM" product from underlying database returned in `MyEditor`.

2. Retrieves the root product

As a next step, the UC retrieves the root product object from the returned editor.

```
...
Dim MyRootOcc As VPMRootOccurrence
Set MyRootOcc = MyEditor.ActiveObject

Dim MyRootProduct As VPMReference
Set MyRootProduct = MyRootOcc.PLMEntity
...
```

3. Retrieves the PartBody

In this step UC retrieves the main body of the 3D shape aggregated by the root product.

```
...
Dim MyContext As PLMProductService
Set MyContext = MyEditor.GetService("PLMProductService")
```

```

Dim MyPart As Part
Dim MyPartInstance As VPMRepInstance
Dim MyBody As Body
For Each MyEntity In MyContext.EditedContent
    Dim MyRef As VPMReference
    Set MyRef = MyEntity

    Dim MyReps As VPMRepInstances
    Set MyReps = MyRef.RepInstances

    For Each MyRep In MyReps
        Dim MyRepRef As VPMRepReference
        Set MyRepRef = MyRep.ReferenceInstanceOf
        Dim attr As String
        attr = MyRepRef.GetAttributeValue("V_discipline")

        If (attr = "Design") Then
            Set MyPart = MyRepRef.GetItem("Part")
            Set MyPartInstance = MyRep
            Set MyBody = MyPart.MainBody
        End If
    Next
Next
...

```

4. Creates the FEM representation and retrieve its root object

In this step UC creates a new FEM representation object. To perform the creation, a simulation representation factory have to be retrieved on the reference object aggregated.

```

...
Dim MyPrdRepFactory As SimPrdRepFactory
Set MyPrdRepFactory = MyRootProduct.GetItem("SimPrdRepFactory")
Dim MyFemRepRef As VPMRepReference
Set MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")
Dim MyFemRepRoot As SimFemRoot
Set MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")

```

The 3D Shape is linked to the FEM representation as assiociated shape.

```

Dim MyProductService As PLMProductService
Set MyProductService = MyEditor.GetService("PLMProductService")

Dim MyReference As Reference
Set MyReference = MyPart.CreateReferenceFromObject(MyBody)
Dim MyLink As AnyObject
Set MyLink = MyProductService.ComposeLink(MyRootOcc, MyPartInstance, MyReference)
MyFemRepRoot.AddAssociatedRep MyLink
...

```

5. Retrieves the mesh set

In this step UC retrieves the mesh set and the collection object of all the mesh parts.

```

...
Dim MyMeshSet As SimMeshSet
Set MyMeshSet = MyFemRepRoot.GetSet("SimNodesElements")
Dim MyMeshParts As SimMeshParts
Set MyMeshParts = MyMeshSet.MeshParts
...

```

6. Creates a Mesh Part

In this step UC creates a surfacic triangle mesh part. The kind of mesher depends of the late type which it's given as argument of the Add method.

```

...
Dim MyMeshPart As SimMeshPart
Set MyMeshPart = MyMeshParts.Add("CATFmtSurfTriaMesher")

```

Then, you should set the mesh part support. A link to the support object must be created first.

```

' Set the Mesh Part support
Set MyPartSupp = MyPart.FindObjectByName("Fill.1")
Set MyReference = MyPart.CreateReferenceFromObject(MyPartSupp)
Set MyLink = MyContext.ComposeLink(MyRootOcc, MyPartInstance, MyReference)
MyMeshPart.AddSupport MyLink

```

Finally, the mesh part attributes can be valuated.

```

'Define the Mesh Part
MyMeshPart.SetAttributeValue "Mesh", "MeshSizeValue", "20 mm"
MyMeshPart.SetAttributeValue "Mesh", "ElementOrder", 2
MyMeshPart.SetAttributeValue "Mesh", "AbsoluteSag", 2
MyMeshPart.SetAttributeValue "Mesh", "AbsoluteSagValue", "2 mm"
...

```

For further informations about the mesher late type and their attributes, please refer to the "Mesh Parts and Their Attributes" article. [\[1\]](#)

7. Creates a local topology specification

In this step UC creates a local topology specification which will be used as support of a mesh specification. As the mesh part creation, the topology specification creation, support definition and attributes valuation.

```

...
Dim MyTopologySpecifications As SimTopologySpecifications
Set MyTopologySpecifications = MyMeshPart.GetTopologySpecifications
Dim MyTopologySpecification As SimTopologySpecification
Set MyTopologySpecification = MyTopologySpecifications.Add("CATFmtExternalCurveLocalSpec")

```

```

'Set the mesh specification support
Set MySpecSupp = MyPart.FindObjectByName("MyLine")
Set MyReference = MyPart.CreateReferenceFromObject(MySpecSupp)
Set MyLink = MyContext.ComposeLink(MyRootOcc, MyPartInstance, MyReference)
Dim MyLinkAccess As SimLinkAccess
Set MyLinkAccess = MyTopologySpecification.GetItem("SimLinkAccess")
MyLinkAccess.AddLink "ConnectorList", MyLink

'Define the topology specification
MyTopologySpecification.SetAttributeValue "ToleranceValue", "1 mm"
...

```

For further informations about the topology specification late type and their attributes, please refer to the "Mesh Parts and Their Attributes" article. [1]

8. Creates a local mesh specification

In a manner similar to the one shown above, UC creates a local mesh specification.

```

...
Dim MyMeshSpecifications As SimMeshSpecifications
SetMyMeshSpecifications = MyMeshPart.GetMeshSpecifications
Dim MyMeshSpecification As SimMeshSpecification
SetMyMeshSpecification = MyMeshSpecifications.Add("CATFmtStudioDistributionSpec")

'Set the mesh specification support
SetMyLinkAccess = MyMeshSpecification.GetItem("SimLinkAccess")
MyLinkAccess.AddLink "ConnectorList", MyTopologySpecification

'Define the mesh specification
MyMeshSpecification.SetAttributeValue "Type", 3
MyMeshSpecification.SetAttributeValue "NbElementsValue", 10
MyMeshSpecification.SetAttributeValue "Symmetry", 2
MyMeshSpecification.SetAttributeValue "RatioValue", 3
...

```

For further informations about the mesh specification late type and their attributes, please refer to the "Mesh Parts and Their Attributes" article. [1]

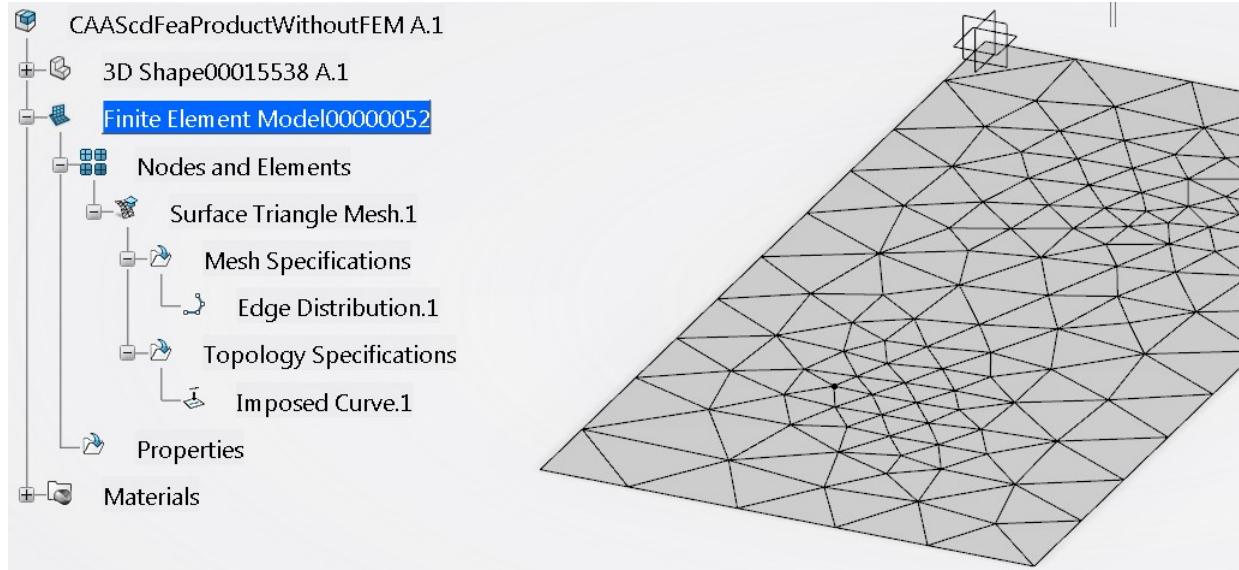
9. Generates the mesh

In this step UC generates the mesh by the update process.

```

...
MyMeshPart.Update
...
```

Fig.1: Generated Mesh



References

[1] [Mesh Part and Their Attributes](#)

Creating Mesh Part

This use case primarily focuses on the methodology to create a Mesh Part.

Before you begin: Note that:

- You should first launch CATIA and import the CAAScdfeaProductWithoutFEM.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaModeling\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- You should open the product called CAASCDfeaProductWithoutFEM.
- Launch the structural scenario App and create a structural simulation.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaCreateMPWithPythonSource.htm](#)

This use case can be divided in nine steps:

1. [Retrieves the simulation and its product](#)
2. [Retrieves the PartBody](#)
3. [Creates the FEM representation, Retrieves its root object and Associates the PartBody to this](#)
4. [Retrieves the mesh set](#)
5. [Creates a mesh part](#)
6. [Creates a local topology specification](#)
7. [Creates a local mesh specification](#)
8. [Generates the mesh](#)

1. Retrieves the simulation and its product

As a first step, the UC retrieves the simulation and its product from the simulation editor

```
...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...
```

2. Retrieves the PartBody

In this step UC retrieves the main body of the 3D shape aggregated by the simulation's Model.

```
...
myModelRepInstances = myModel.RepInstances

for myPartRepInstance in myModelRepInstances:
    myPartInstanceRef = myPartRepInstance.ReferenceInstanceOf
    attr = myPartInstanceRef.GetAttributeValue("V_discipline")
    if attr == "Design":
        myPart = myPartInstanceRef.GetItem("Part")
        myPartInstance = myPartRepInstance
        myBody = myPart.MainBody
...

```

3. Creates the FEM representation, Retrieves its root object and Associates the PartBody to this

In this step UC creates a new FEM representation object.

To perform the creation, a simulation representation factory has to be retrieved on the reference object which will aggregate the FEM representation.

```
...
MyPrdRepFactory = myModel.GetItem("SimPrdRepFactory")
MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")
MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")
...
```

Now UC appends the main body previously retrieved to the associated shapes of the FEM representation.

First, the UC opens the model and retrieves the product editor and the root occurrence

```
...
oOpenService = CATIA.GetService("PLMOpenService")
oOpenService.PLMOpen (myModel)
myProductEditor = CATIA.ActiveEditor
myRootOcc = myProductEditor.ActiveObject
```

Now it creates the link to associate with the FEM

```
MyReference = myPart.CreateReferenceFromObject(myBody)
myProductService = myProductEditor.GetService("PLMProductService")
MyLink = myProdService.ComposeLink(myRootOcc, myPartInstance, MyReference)
MyFemRepRoot.AddAssociatedRep (MyLink)
...
```

4. Retrieves the mesh set

In this step UC retrieves the mesh set and the collection object of all the mesh parts.

```
...
myMeshSet = MyFemRepRoot.GetSet("SimNodesElements")
myMeshParts = myMeshSet.MeshParts
...
```

5. Creates a mesh part

In this step UC creates a surfacic triangle mesh part. The kind of mesher depends of the late type which it's given as argument.

```
...
myMeshPart = myMeshParts.Add("CATFmtSurfTriaMesher")
...
```

Then, you should set the mesh part support. A link to the support object must be created first.

```
...
MyPartSupp = myPart.FindObjectByName("Fill.1")
MyReference = myPart.CreateReferenceFromObject(MyPartSupp)
MyLink = myProdService.ComposeLink(myRootOcc, myPartInstance, MyReference)
myMeshPart.AddSupport (MyLink)
...
```

Finally, the mesh part attributes can be evaluated.

```
...
myMeshPart.SetAttributeValue("Mesh", "MeshSizeValue", "20 mm")
myMeshPart.SetAttributeValue("Mesh", "ElementOrder", 2)
myMeshPart.SetAttributeValue("Mesh", "AbsoluteSag", 2)
myMeshPart.SetAttributeValue("Mesh", "AbsoluteSagValue", "2 mm")
...
```

For further informations about the mesher late type and their attributes, please refer to the "Mesh Parts and Their Attributes" article. [1]

6. Creates a local topology specification

In this step UC creates a local topology specification which will be used as support of a mesh specification. As the mesh part creation, the topology specification creation, support definition and attributes valuation.

```
...
MyTopologySpecifications = myMeshPart.GetTopologySpecifications()
MyTopologySpecification = MyTopologySpecifications.Add("CATFmtExternalCurveLocalSpec")

MySpecSupp = myPart.FindObjectByName("MyLine")
MyReference = myPart.CreateReferenceFromObject(MySpecSupp)
MyLink = myProductService.ComposeLink(myRootOCC, myPartInstance, MyReference)
MyLinkAccess = MyTopologySpecification.GetItem("SimLinkAccess")
MyLinkAccess.AddLink("ConnectorList", MyLink)

MyTopologySpecification.SetAttributeValue("ToleranceValue", "1 mm")
```

For further informations about the topology specification late type and their attributes, please refer to the "Mesh Parts and Their Attributes" article. [1]

7. Creates a local mesh specification

In a manner similar to the one shown above, UC creates a local mesh specification.

```
...
MyMeshSpecifications = myMeshPart.GetMeshSpecifications()
MyMeshSpecification = MyMeshSpecifications.Add("CATFmtStudioDistributionSpec")

MyLinkAccess = MyMeshSpecification.GetItem("SimLinkAccess")
MyLinkAccess.AddLink("ConnectorList", MyTopologySpecification)

MyMeshSpecification.SetAttributeValue("Type", 3)
MyMeshSpecification.SetAttributeValue("NbElementsValue", 10)
MyMeshSpecification.SetAttributeValue("Symmetry", 2)
MyMeshSpecification.SetAttributeValue("RatioValue", 3)
...
```

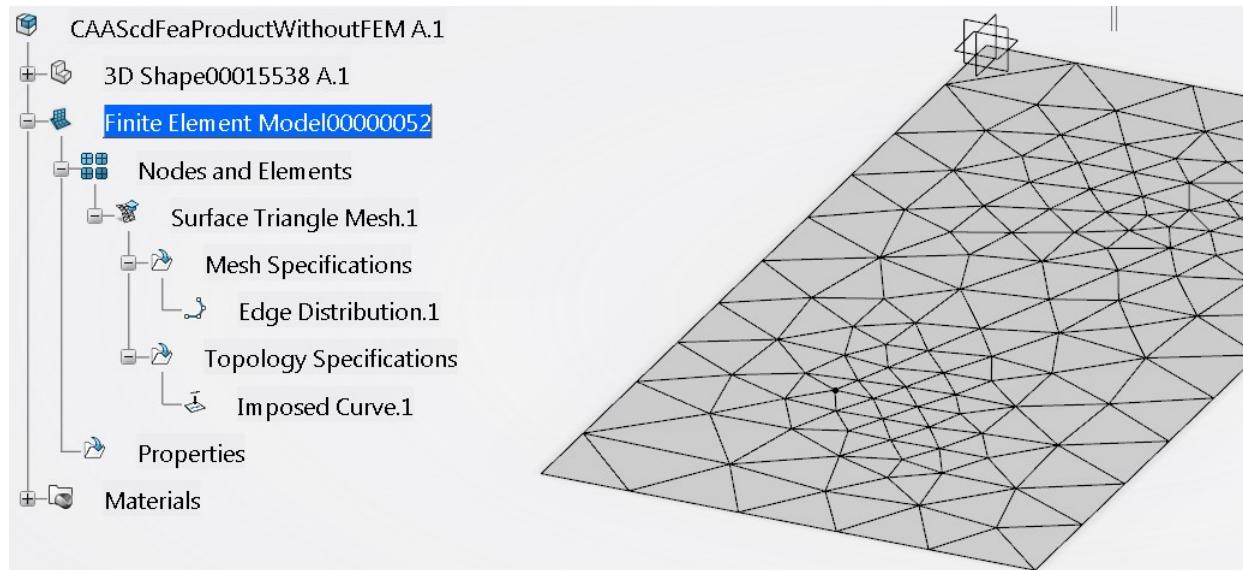
For further informations about the mesh specification late type and their attributes, please refer to the "Mesh Parts and Their Attributes" article. [1]

8. Generates the mesh

In this step UC generates the mesh by the update process.

```
...
myMeshPart.Update
...
```

Fig.1: Generated Mesh



References

[1] [Mesh Part and Their Attributes](#)

Beam Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **One Node Beam Mesh Part** attributes and its local specifications.

Beam Mesh Part Creation**Beam Mesh Part Support Definition:****Global Mesh Specifications:**

- [MeshSizeValue](#)
- [ElementOrder](#)
- [AbsoluteSag](#)
- [AbsoluteSagValue](#)
- [AutomaticMeshCapture](#)
- [AutomaticMeshCaptureTolValue](#)

Global Topology Specifications:

- [AngleBetweenCurves](#)

Local Topology Specifications:

- [CATFmtExternalPointLocalSpec](#) (Imposed Point):
 - [ToleranceValue](#)
 - [ProjectOnGeometry](#)

Local Mesh 1D Specifications:

- [CATFmtStudioDistributionSpec](#) (Edge Distribution):
 - [Type](#)
 - [Combination](#)
 - [CombinationUnif](#)
 - [NbElementsValue](#)
 - [RatioValue](#)
 - [Size1Value](#)
 - [Size2Value](#)
 - [Symmetry](#)
 - [Reverse](#)

Beam Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Beam mesh part, the late type is "CATFmtBeamRulesMesher"

The Beam mesh part can be created by using **Add** method of **SimMeshParts** object.

Beam Mesh Part Support Definition

Beam Mesh Part support can be set using following method of the **SimMeshPart** object:

- **AddSupport** (*iSupport As AnyObject*) to add a new support to the mesh part

Global Mesh Specifications

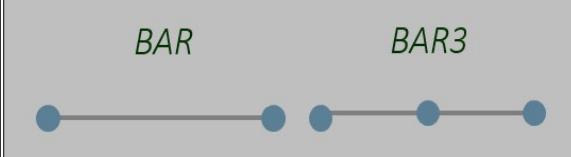
Global mesh specifications can be set or retrieved using following methods of the **SimMeshPart** object:

- **SetAttributeValue** (*iSetType As String*, *iAttribute As String*, *iValue*) to set a mesh attribute where:
 - *iSetType*="Mesh"
 - *iAttribute* is the attribute's name
 - *iValue* is the attribute's value
- **GetAttributeValue** (*iSetType As String*, *iAttribute As String*, *oValue*) to get a mesh attribute where:
 - *iSetType*="Mesh"
 - *iAttribute* is the attribute's name
 - *oValue* is the attribute's value

MeshSizeValue

type: <i>double</i> valued to mesh elements size. default value: 10.	Attribute which specifies the mesh element target size that the mesher tries to respect.
---	--

ElementOrder

type: <i>integer</i> valued to: • 1: to get a mesh with linear elements • 2: to get a mesh with quadratic elements default value: 1	Attribute which specifies the number of intermediate nodes per element edge, either linear BAR (on the left picture) or quadratic BAR3 (on the right picture).	
--	--	--

AbsoluteSag

type: <i>integer</i> valued to:	
--	--

<ul style="list-style-type: none"> 1: to create a mesh that does not respect sag 2: to create a mesh that respects sag <p>default value: 1</p>	<p>Attribute which specifies if the mesh is created with a maximum imposed sag.</p>	
---	---	--

AbsoluteSagValue

<p>type: <i>double</i> valued to the target maximum sag when AbsoluteSag attribute is valued to 2.</p> <p>default value: 1.0</p>	<p>Attribute which specifies the maximum value of the absolute sag (distance between the geometry of the mesh).</p> <p>WARNING: It is taken into account only if the AbsoluteSag attribute is equal to 2</p>
---	--

AutomaticMeshCapture

<p>type: <i>integer</i> valued to:</p> <ul style="list-style-type: none"> 1: to ignore automatic mesh capture 2: to compute automatic mesh capture <p>default value: 1</p>	<p>Attribute which specifies if external meshes must be captured. This attribute is linked to the attribute AutomaticMeshCaptureTolValue.</p>
--	--

AutomaticMeshCaptureTolValue

<p>type: <i>double</i> valued to the maximum distance under which mesh is captured.</p> <p>default value: 0.</p>	<p>Attribute which specifies the maximum distance for mesh capture: if a mesh (element edges) is farthest from the support than this tolerance, it will not be captured even if the AutomaticMeshCapture attribute is valued to 2.</p> <p>WARNING: to precise exactly the list of mesh parts to capture, the method SetMeshPartsToCapture of SimMeshPart object can be used:</p> <ul style="list-style-type: none"> • SetMeshPartsToCapture () <p>To retrieve the parents mesh parts another method of SimMeshPart object can be used:</p> <ul style="list-style-type: none"> • Parent ()
--	--

Global Topology Specifications

Global topology specifications can be set or retrieved using following methods:

- **SetAttributeValue** (*iSetType As String, iAttribute As String, iValue*) to set a topology attribute
- **GetAttributeValue** (*iSetType As String, iAttribute As String, oValue*) to get a topology attribute

iType is the type of specification and **iType = "Topology"** for Global Topology Specifications.

AngleBetweenCurves

<p>type: <i>double (radian unit)</i> valued to the angle under which vertices are deactivated</p> <p>default value: 20°</p>	<p>Attribute which specifies the maximum angle under which a vertex connected exactly to two constrained edges can be deactivated.</p>	
---	--	--

Local Topology Specifications**CATFmtExternalPointLocalSpec**

<p>Specification attributes are managed using SimTopologySpecification interface:</p> <ul style="list-style-type: none"> • Geometric attribute (supports) <p>Other attributes are managed using (GetAttributeValue, SetAttributeValue):</p> <ul style="list-style-type: none"> • ToleranceValue: <i>double</i> valued to the maximum distance between external point and its projection on geometry to mesh. <p>default value: 0.</p> <ul style="list-style-type: none"> • ProjectOnGeometry: <i>integer</i> valued to the target geometry choice <p>default value: 1</p>	<p>Description: This specification is used in order to project external points (points that don't belong to the geometry to mesh) in order to constrain the mesh. External points definition (supports) and projection tolerance are mandatory. The overall behavior is the following:</p> <ul style="list-style-type: none"> • if the distance between an external point and the geometry to mesh is greater than the ToleranceValue, the point will not be projected • if the distance between an external point and the geometry to mesh is lower than the ToleranceValue, the point will be projected. The final location of this point depends on ProjectOnGeometry attribute: <ul style="list-style-type: none"> ◦ if ProjectOnGeometry is equal to 1, the node representing the point will be located exactly on the projection point on the surface geometry ◦ if ProjectOnGeometry is equal to 2, the node representing the point projection will be located exactly on the external point (same Cartesian coordinates)
---	--

Local Mesh 1D Specifications

WARNING: 1D grouping specifications are the only supports possible for Local Mesh 1D Specifications. Before creating a Local Mesh 1D Specification, it is mandatory to create at least one 1D Grouping Specification with the targeted geometry supports.

CATFmtStudioDistributionSpec

Distribution specification (named for example *localSpec*) can be applied only on **one edge** resulting of an unique grouping specification. Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- **Type: integer** defining the type of distribution valued to:
 - 1: for a Uniform distribution (Isometric)
 - 2: for an Arithmetic distribution
 - 3: for a Geometric distribution
 - 4: for a distribution defined by a Law

default value: 1

- **Combination: integer** indicating what are the possible combination for Arithmetic or Geometric distributions valued to:
 - 1: for a distribution defined by the number of elements (*NbElements*) and the size ratio of the elements (*Ratio21*)
 - 2: for a distribution defined by the number of elements (*NbElements*) and the size (*Size1*) at first vertex
 - 3: for a distribution defined by the number of elements (*NbElements*) and the size (*Size2*) at second vertex
 - 4: for a distribution defined by the size ratio (*Ratio21*) and the size (*Size1*) at first vertex
 - 5: for a distribution defined by the size ratio (*Ratio21*) and the size (*Size2*) at second vertex
 - 6: for a distribution defined by the size (*Size1*) at first vertex and the size (*Size2*) at second vertex

default value: 1

- **CombinationUnit: integer** indicating what are the possible combination for Uniform distributions valued to:
 - 1: for a distribution defined by the number of elements (*NbElements*)
 - 2: for a distribution defined by the size (*Size1*) of the elements

default value: 1

- **NbElementsValue: integer** defining the number of elements (*NbElements*)

default value: 4

- **RatioValue: double** defining the ratio between the max element size and the min element size (*Ratio21*)

default value: 1.

- **Size1Value: double** defining the size at first vertex (*Size1*)

default value: 0.

- **Size2Value: double** defining the size at second vertex (*Size2*)

default value: 0.

- **Symmetry: integer** valued to:
 - 1: to keep the distribution unchanged
 - 2: to compute a symmetric distribution with current attributes

default value: 1

- **Reverse: integer** usable for Law and non Uniform distribution valued to:
 - 1: to keep the distribution unchanged
 - 2: to invert the distribution

default value: 1

Description: This specification creates a node distribution on an edge resulting of a 1D grouping specification. Depending on the choice of the distribution type (Type attribute), Combination or CombinationUnif attributes several various distributions "Option" can be computed. The next table summarize all "Option":

- Isometric (Uniform) distribution is defined by only one attributes(Option 1 and 2):
 - *Nbelements* (**CombinationUnif= 1**)
 - *Size1* (**CombinationUnif= 2**)
- Non isometric distribution is defined by two attributes as indicated in the next table. There are six couples of attributes (Option 3 to 8):
 - *Nbelements* and *Ratio21* (**Combination = 1**)
 - *NbElements* and *Size1* (**Combination = 2**)
 - *NbElements* and *Size2* (**Combination = 3**)
 - *Ratio21* and *Size1* (**Combination = 4**)
 - *Ratio21* and *Size2* (**Combination = 5**)
 - *Size1* and *Size2* (**Combination = 6**)

	<i>"Isometric"</i>		<i>"Arithmetric"\ "Geometric"</i>					
<i>"Option"</i>	1	2	3	4	5	6	7	8
<i>"NbElements"</i>	●		●	●	●			
<i>"Ratio21"</i>			●				●	●
<i>"Size1"</i>		●		●		●	●	●
<i>"Size2"</i>				●	●	●	●	●

The next table shows examples of various distributions using additional Symmetry and Law attributes.

Examples	1	2	3	4	5
Attributes	Isometric	Arithmetric	Geometric	Uniform	Law
Type	«Arithmetric»	«Geometric»	«Uniform»	«Law»	
NbElements	●		10	●	●
Symmetric	●		Yes	●	●
Size1	50	●		15	●
Size2	200	●		●	●
Ratio21	●	2		●	●
Reverse	●	●		●	●
Law	●	●	●	●	$y=x^2$

History

Version: 1 [December 2016] Document created

Coating 1D Mesh Parts Attributes

Technical Article

Abstract

This article presents an overview of the **Coating 1D Mesh Part** attributes and its local specifications.

[Coating 1D Mesh Part Creation](#)

[Coating 1D Mesh Part Support Definition](#)

[The Mesh Attributes:](#)

- [ExtractionType](#)

[The Local Specifications:](#)

- [CATFmtCoatingLocalSpecification](#)

Coating 1D Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Coating 1D Mesh Part, the late type is "**CATFmtCoating1DMesher**".

The Coating 1D Mesh Part can be created by using **Add** method of **SimMeshParts** object.

Coating 1D Mesh Part Support Definition

The Coating 1D Mesh Part support can be initialized and managed using the **AddSupport** method of **SimMeshPart** object:

- **AddSupport (iSupport As AnyObject)** to add a new support to the mesh part

Mesh attributes of the Coating 1D mesh part

The attributes referenced in this section drive the **Mesh** specification of Coating 2D mesh and define precisely what type of mesh is wanted by user.

Mesh specifications can be set or retrieved using the type "**Mesh**" on the **SetAttributeValue** and **GetAttributeValue** services of the **SimMeshPart** object.

ExtractionType

Type: **int**

Valuated to

- 1 to generate a 1D element for each edge of the 2D meshes (**All Edges**).
- 2 to generate a 1D element for each boundary edge of the 2D meshes (**Boundary Edges**).
- 3 to generate a 1D element for each internal edge of the 2D meshes (**Internal Edges**).
- 4 to generate a 1D element for each constrained edge of the 2D meshes (**Constrained Edges**).
- 5 to generate a 1D element for each unconstrained edge of the 2D meshes (**Non Constrained Edges**).
- 6 to generate a 1D element only for local specifications (**None**).

This attribute specifies which edges should be extracted from the support.

Default Value: 1

Local specifications of the Coating 1D mesh part

The Coating 1D Mesh Part has only one kind of local specifications: **mesh** local specifications. These specifications are managed by collection object:

- **SimMeshSpecifications:** for the collection of local mesh specifications

You should use the following methods of **SimMeshPart** object to retrieve these collections:

- **GetMeshSpecifications** to retrieve the local mesh specification collection

Local specifications are managed from the collection objects:

- **Add (iType As String)** to create a new local specification with:
 - iType is the late type of the specification to create
- **Item (iIndex)** to retrieve a local specification with:
 - iIndex is the index of the local specification in its collection
- **Remove (iIndex)** to delete a local specification with:
 - iIndex is the index of the local specification in its collection

Attribute of local specification are set using following method from **SimMeshSpecification** or **SimTopologySpecification** object depending of the kind of local specification:

- **SetAttributeValue (iAttribute As String, iValue)** to set a local specification attribute value

In a manner similar, attribute of local specification are retrieved using following method:

- **GetAttributeValue (iAttribute As String, iValue)** to retrieve a local specification

Geometric attribute of local specification (supports) are managed using **SimLinkAccess** object:

- **AddLink** (*iName As String, iLink As AnyObject*)
- **RemoveAllLinks** (*iName As String*)
- **RemoveLink** (*iName As String, iLink As AnyObject*)
 - *iName* is the name of the link container, here it's "**ConnectorList**"

CATFmtCoatingLocalSpecification

This specification specifies the edges which are to be included or excluded while computing the coating mesh. You can specify multiple edges to one local specification. If you want to include some edges, and exclude some other then you need to create at least two local specifications. All the edges which are to be included, can be added as support to one local specification with the attribute "LocalExtractionType" valued to 1. Similarly edges which are to be exclude can be added as support to other local specification with the attribute "LocalExtractionType" valued to 2.

LocalExtractionType

Type: **int**

Valuated to

- 1 to include the edges defined as support of the local specification
- 2 to exclude the edges defined as support of the local specification

This attribute specifies if the edges defined as support of the local specification should be included (1) or excluded (2).

Default Value: 1

History

Version: 1 [August 2018] Document created

Coating 2D Mesh Parts Attributes

Technical Article

Abstract

This article presents an overview of the **Coating 2D Mesh Part** attributes and its local specifications.

[Coating 2D Mesh Part Creation](#)

[Coating 2D Mesh Part Support Definition](#)

[The Mesh Attributes:](#)

- [ExtractionType](#)

[The Local Specifications:](#)

- [CATFmtCoatingLocalSpecification](#)

Coating 2D Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Coating 2D Mesh Part, the late type is "**CATFmtCoating2DMesher**".

The Coating 2D Mesh Part can be created by using **Add** method of **SimMeshParts** object.

Coating 2D Mesh Part Support Definition

The Coating 2D Mesh Part support can be initialized and managed using the **AddSupport** method of **SimMeshPart** object:

- **AddSupport** (*iSupport As AnyObject*) to add a new support to the mesh part

Mesh attributes of the Coating 2D mesh part

The attributes referenced in this section drive the **Mesh** specification of Coating 2D mesh and define precisely what type of mesh is wanted by user.

Mesh specifications can be set or retrieved using the type "**Mesh**" on the **SetAttributeValue** and **GetAttributeValue** services of the **SimMeshPart** object.

ExtractionType

Type: **int**

Valuated to

- 1 to extract all the boundary faces This attribute specifies if the support boundary faces should be extracted from the support mesh (1) or not (2).
- 2 to don't extract any face

Default Value: 1

Local specifications of the Coating 2D mesh part

The Coating 2D Mesh Part has only one kind of local specifications: **mesh** local specifications. These specifications are managed by collection object:

- **SimMeshSpecifications**: for the collection of local mesh specifications

You should use the following methods of **SimMeshPart** object to retrieve these collections:

- **GetMeshSpecifications** to retrieve the local mesh specification collection

Local specifications are managed from the collection objects:

- **Add** (*iType As String*) to create a new local specification with:
 - *iType* is the late type of the specification to create
- **Item** (*iIndex*) to retrieve a local specification with:
 - *iIndex* is the index of the local specification in its collection
- **Remove** (*iIndex*) to delete a local specification with:
 - *iIndex* is the index of the local specification in its collection

Attribute of local specification are set using following method from **SimMeshSpecification** or **SimTopologySpecification** object depending of the kind of local specification:

- **SetAttributeValue** (*iAttribute As String, iValue*) to set a local specification attribute value

In a manner similar, attribute of local specification are retrieved using following method:

- **GetAttributeValue** (*iAttribute As String, iValue*) to retrieve a local specification

Geometric attribute of local specification (supports) are managed using **SimLinkAccess** object:

- **AddLink** (*iName As String, iLink As AnyObject*)
- **RemoveAllLinks** (*iName As String*)
- **RemoveLink** (*iName As String, iLink As AnyObject*)
 - *iName* is the name of the link container, here it's "**ConnectorList**"

CATFmtCoatingLocalSpecification

This specification specifies the faces which are to be included or excluded while computing the coating mesh. You can specify multiple faces to one local specification. If you want to include some faces, and exclude some other then you need to create at least two local specifications. All the faces which are to be included, can be added as support to one local specification with the attribute "LocalExtractionType" valued to 1. Similarly faces which are to be exclude can be added as support to other local specification with the attribute "LocalExtractionType" valued to 2.

LocalExtractionType

Type: **int**

Valuated to

- 1 to include the faces defined as support of the local specification
- 2 to exclude the faces defined as support of the local specification

This attribute specifies if the faces defined as support of the local specification should be included (1) or excluded (2).

Default Value: 1

History

Version: 1 [August 2018] Document created

Hex-Dominant Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Hex-Dominant Mesh Part** attributes and its local specifications.

[Hex-Dominant Mesh Part Creation](#)

[Hex-Dominant Mesh Part Support Definition](#)

[Global Mesh Specifications](#)

- [Size](#)
- [MinSize](#)
- [Angle](#)
- [UseBoundaryLayers](#)
- [Size1Value](#)
- [UseNbElements](#)
- [NbElementsValue](#)
- [Splitting](#)

[Local Mesh Specifications](#)

- [CATFmtHexLocalMeshSizeFace](#)
- [CATFmtHexLocalMeshSizeBox](#)

Hex-Dominant Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the hex-dominant mesh part, the late type is "**CATFmtHexMesher**".

The hex-dominant mesh part can be created by using **Add** method of **SimMeshParts** object.

Hex-Dominant Mesh Part Support Definition

The Hex-Dominant Mesh Part support can be initialized and managed using the **AddSupport** method of **SimMeshPart** object:

- **AddSupport (iSupport As AnyObject)** to add a new support to the mesh part

Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods of **SimMeshPart** object:

- **SetAttributeValue (iSetType As String, iAttribute As String, iValue)** to set a mesh attribute
- **GetAttributeValue (iSetType As String, iAttribute As String, oValue)** to get a mesh attribute value

iAttribute is the name of specification and **iSetType = "Mesh"** for all Global Mesh Specifications.

Size

Type: **double**

This attribute specifies the maximum mesh element size that the mesher tries to respect.

Valuated to mesh elements maximum target size.

MinSize

Type: **double**

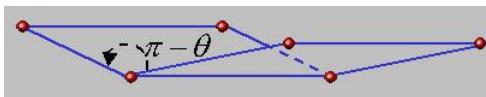
This attribute specifies the minimum mesh element size that the mesher tries to respect.

Valuated to mesh elements minimum target size.

Angle

Type: **double** (radians)

This attribute specifies the smallest angle θ between adjacent faces that will result in an edge between mesh element faces.



UseBoundaryLayers

Type: **int**

Valuated to

This attribute specifies whether boundary layers should be added to the mesh (2) or not (1).

- 1 to produce a mesh without boundary layers
- 2 to produce a mesh with boundary layers

Size1Value

Type: **double**

This attribute specifies the thickness of the first layer of elements that the mesher tries to respect along the boundary.

Valuated the expected thickness for the first layer. **Warning:** this attribute is only taken into account if **UseBoundaryLayers** is set to 2.

UseNbElements

Type: **int**

Valuated to

This attribute specified if the number of boundary layers is an user input (2) or not (1). If it is set to 1, the mesher will compute the number of boundary layers automatically based on the mesh size to ensure smooth transition towards the interior of the volume.

Warning: this attribute is only taken into account if **UseBoundaryLayers** is set to 2.

NbElementsValue

Type: **int**

This attribute specified the number of boundary layers to be generated along the boundary.

Valuated to the expected number of boundary layers. **Warning:** this attribute is only taken into account if **UseBoundaryLayers** and **UseNbElements** are both set to 2.

Splitting

Type: **int**

Valuated to

This attribute specified which method is used to manage edge capture while creating a hex-dominant mesh.

- 1 to use the default method which provides better accuracy in capturing sharp edges.
- 2 to use the alternative method, the "Standard edge capture", which is typically more robust when dealing with complex, dirty geometry.

Local Mesh Specifications

Local mesh specifications are created by the **Add** method of the **SimMeshSpecifications** object.

- **Add (iType As String)** with:
 - **iType** is the late type of the mesh specification to create.

The **Item** and the **Remove** methods work with an index which is the position of the specification in the collection. The position index starts from 1.

The support of the local topology specification is managed by the **SimLinkAccess** object:

- **AddLink** (*iName* As String, *iLink* As AnyObject)
- **RemoveAllLinks** (*iName* As String)
- **RemoveLink** (*iName* As String, *iLink* As AnyObject)
 - *iName* is the name of the link container, here it's "ConnectorList"

The specification values are managed by the following methods of **SimMeshSpecification**:

- **GetAttributeValue** (*iAttribute* As String, *oValue*)
- **SetAttributeValue** (*iAttribute* As String, *iValue*)
 - *iAttribute* is the attribute name to set/retrieve.

CATFmtHexLocalMeshSizeFace

The size to be applied on the support faces is specified through **GetAttributeValue** and **SetAttributeValue** with *iName* = "SizeValue" (type **double**).

This specification is used to refine the mesh along a given set of faces.

The target size should be lower than the maximum mesh size given through the global **Size** attribute.

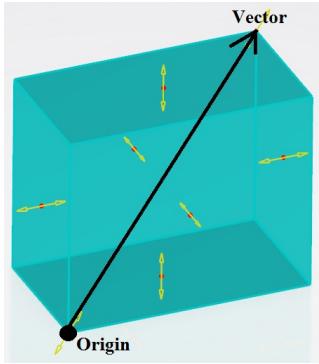
CATFmtHexLocalMeshSizeBox

All specification attributes are specified through **GetAttribute** and **SetAttribute**:

This specification is used to refine the mesh inside a given Cartesian box: the box is defined by its lower left corner and a vector transforming it into the upper right corner, as shown on the picture below.

The target size should be lower than the maximum mesh size given through the global **Size** attribute.

- **OriginX** (type **double**): the X coordinate of the lower left corner of the box
- **OriginY** (type **double**): the Y coordinate of the lower left corner of the box
- **OriginZ** (type **double**): the Z coordinate of the lower left corner of the box
- **Vector0X** (type **double**): the dimension of the box along the X axis
- **Vector1Y** (type **double**): the dimension of the box along the Y axis
- **Vector2Z** (type **double**): the dimension of the box along the Z axis
- **SizeValue** (type **double**): the target mesh size inside the box



Octree Mesh Parts Attributes

Technical Article

Abstract

This article presents an overview of the Octree Mesh Parts attributes and their local specifications.

[Octree Mesh Part Creation](#)

[Octree Mesh Part Support Definition](#)

[The Mesh Attributes:](#)

- [MeshSizeValue](#)
- [ElementOrder](#)
- [AbsoluteSag](#)
- [AbsoluteSagValue](#)
- [ProportionalSag](#)
- [ProportionalSagValue](#)
- [MinMeshSize](#)
- [QualityCriteria](#)
- [InteriorSize](#)
- [InteriorSizeValue](#)
- [MeshSimplification](#)
- [MeshSimplificationValue](#)
- [MinJacobian](#)
- [MinJacobianValue](#)
- [GlobalSplit](#)

[The Local Specifications:](#)

- [CATFmtLocalSpecLocalSize](#)
- [CATFmtLocalMeshDistribution](#)
- [CATFmtImposedGeometries](#)
- [CATFmtPreserveGeometries](#)

Octree Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Octree Mesh Part, the late type is "**CATFmtOctree3DRulesMesher**" for a tri-dimensional mesh (to mesh a solid with tetrahedron elements) and "**CATFmtOctree2DRulesMesher**" for a surface mesh (to mesh a surface with triangle elements).

The Octree Mesh Part can be created by using **Add** method of **SimMeshParts** object.

Octree Mesh Part Support Definition

The Octree Mesh Part support can be initialized and managed using the **AddSupport** method of **SimMeshPart** object:

- **AddSupport (iSupport As AnyObject)** to add a new support to the mesh part

Mesh attributes of the Octree mesh part

The attributes referenced in this section drive the **Mesh** specification of Octree mesh and define precisely what type of mesh is wanted by user.

Mesh specifications can be set or retrieved using the type "**Mesh**" on the **SetAttributeValue** and **GetAttributeValue** services of the **SimMeshPart** object.

MeshSizeValue

type: this argument is a double valued to the wanted size for mesh elements (it is an objective).	Attribute which specifies the global size of the mesh (it's an objective, Octree mesher tried to approach nearest possible from this size). WARNING: the size value is the objective size for "skin" mesh, for 3-dimensional mesh user must give an interior size, else 3D Octree mesher tried to create interior element as biggest as possible.
---	--

ElementOrder

type: this argument is an integer valued to, • 1 if user wants to mesh with linear elements (<i>TR3 or TR4 depending from the type of Octree mesh part: 2D or 3D</i>); • 2 if user wants to mesh with parabolic elements (<i>TR6 or TR10</i>).	Attribute which specifies the degree of elements that user wants to create: there is two possibilities, linear elements (TR3 or TR4 depending from the Octree mesh part type (2D or 3D)) and parabolic elements (TR6 or TR10).	
--	--	--

AbsoluteSag

type: this argument is an integer valued to, • 1 if user doesn't want to apply mesh with absolute sag; • 2 if user wants to apply mesh with absolute sag.	Attribute which specifies if the mesh is created with an imposed absolute minimum sag. This attribute is linked to the AbsoluteSagValue attribute.
---	--

AbsoluteSagValue

type: this argument is a double valued to the wanted proportional sag.	Attribute which specifies the size of the maximum absolute sag accepted, this attribute is active when the "AbsoluteSag" attribute is active only. The maximum size of the sag is imposed and contrary to the proportional sag this one doesn't take in account the "local" size of the mesh: in other terms if the curvature radius of the geometry is big, the imposed sag can imply a very small size for the mesh. <i>Example:</i> in the right picture the curvature radius is biggest in the first case, that's why, in order to satisfy the imposed sag condition, the size of the mesh is smaller than in the second case. This attribute is used only when the AbsoluteSag attribute is active.	
--	--	--

ProportionalSag

type: this argument is an integer valued to, • 1 if user doesn't want to apply mesh with proportional sag; • 2 if user wants to apply mesh with proportional sag.	Attribute which specifies if the mesh is created with an imposed proportional minimum sag. This attribute is linked to the ProportionalSagValue attribute.
---	--

ProportionalSagValue

	Attribute which specifies the size of the maximum relative	
--	--	--

<p>type: this argument is a double valued to the wanted proportional sag.</p>	<p>sag accepted, this attribute is active when the "ProportionalSag" attribute is active only. The relative sag is calculated by the quotient between the Sag and the length of the edge of a mesh element: Sag/Length.</p> <p>This attribute is used only when the ProportionalSag attribute is active.</p>	
---	---	--

MinMeshSize

<p>type: this argument is a double valued to the minimum size for elements even if the sag required a smaller size for mesh.</p>	<p>Attribute which specifies the minimum size for mesh. An "objective" size for mesh is specified (MeshSizeValue attribute) but the constrain sag required may implies the decrease of the mesh size in order to satisfy it's condition, that's why a minimum size condition exist even if the Octree mesher tries to approach the global mesh size specified.</p>
--	--

QualityCriteria

<p>type: this argument is an integer valued to,</p> <ul style="list-style-type: none"> • 1 if user wants to respect "Skewness" factor for the mesh; • 2 if user wants to respect "Stretch" factor for the mesh; • 3 if user wants to respect "Shape" factor for the mesh. 	<p>Attribute which specifies the factor that user wants to respect for the elements of the mesh. Octree mesher provide the possibility for user to control one of these arguments.</p>	
--	--	--

InteriorSize

<p>type: this argument is an integer valued to,</p> <ul style="list-style-type: none"> • 1 if user doesn't want to apply an imposed maximum interior size for mesh; • 2 if user wants to apply an maximum imposed interior size for mesh. <p>WARNING: this attribute is active when the part is an octree 3D only (because interior size has no sense for surface mesh).</p>	<p>Attribute which specifies if user wants to apply an imposed maximum size for the interior size of the 3D mesh; because, by default, Octree mesher tried to mesh the interior of the solid as biggest as possible. This attribute provide to user the possibility to limit this expansion by imposing a maximum size. <i>Example:</i> On the right example, the first picture represent the mesh without imposed interior size contrary to the second one where the interior size was imposed at the same size than the global mesh size (the mesh is more regular).</p>	
---	--	--

InteriorSizeValue

<p>type: this argument is a double valued to the wanted proportional sag.</p>	<p>Attributes which specifies the value of the imposed interior size for 3D mesh when the InteriorSize attribute is active.</p>
---	---

MeshSimplification

<p>type: this argument is an integer valued to,</p> <ul style="list-style-type: none"> • 1 if user doesn't want to apply a minimum size for mesh elements; • 2 if user wants to apply a minimum size for mesh elements. 	<p>Attribute which specifies the minimum size for a mesh edge, in order to optimize the regularity of the mesh, user can apply the suppression of small mesh elements by activation of this attribute.</p> <p>The right example show the difference between an octree mesh without mesh edges suppression (the highest picture) and with the suppression of smallest mesh edges.</p>	
---	--	--

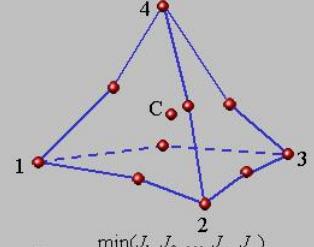
MeshSimplificationValue

type: this argument is a <i>double</i> valued to the wanted mesh simplification.	Attributes which specifies the value of the minimum size for mesh elements when the MeshSimplification attribute is active.
---	---

MinJacobian

Warning: This attribute is available only for parabolic tetrahedron elements (TE10). type: this argument is an <i>integer</i> valued to, <ul style="list-style-type: none"> • 1 if user doesn't want to project intermediate mesh nodes on geometry; • 2 if user wants to project intermediate mesh nodes on geometry. 	Attribute which specifies if the intermediate mesh nodes are created on the geometry. This attribute is linked to the MinJacobianValue attribute.
---	---

MinJacobianValue

type: this argument is a <i>double</i> valued to the wanted minimum size for Jacobian value for which the intermediate nodes is on geometry.	Attribute which specifies the minimum Jacobian accepted for each parabolic element of the mesh when the MinJacobian attribute is active.	 $Q = \frac{\min(J_1, J_2, \dots, J_n, J_c)}{\max(J_1 , J_2 , \dots, J_n , J_c)}$
---	--	---

GlobalSplit

type: this argument is an <i>integer</i> valued to, <ul style="list-style-type: none"> • 1 if user doesn't want ensure at least two layer elements in the thickness of thin geometry; • 2 if user wants ensure at least two layer elements in the thickness of thin geometry; <p>This Attribute concerns only tri-dimensional mesh (to mesh a solid with tetrahedron elements)</p>	Attribute which specifies if the minimum of two mesh elements layers in thin geometries.
--	--

Local specifications of the Octree mesh part

The Octree Mesh Part has only one kind of local specifications: **mesh** local specifications. These specifications are managed by collection object:

- **SimMeshSpecifications:** for the collection of local mesh specifications

You should use the following methods of **SimMeshPart** object to retrieve these collections:

- **GetMeshSpecifications** to retrieve the local mesh specification collection

Local specifications are managed from the collection objects:

- **Add (iType As String)** to create a new local specification with:
 - iType is the late type of the specification to create
- **Item (iIndex)** to retrieve a local specification with:
 - iIndex is the index of the local specification in its collection
- **Remove (iIndex)** to delete a local specification with:
 - iIndex is the index of the local specification in its collection

Attribute of local specification are set using following method from **SimMeshSpecification** or **SimTopologySpecification** object depending of the kind of local specification:

- **SetAttributeValue (iAttribute As String, iValue)** to set a local specification attribute value

In a manner similar, attribute of local specification are retrieved using following method:

- **GetAttributeValue (iAttribute As String, iValue)** to retrieve a local specification

Geometric attribute of local specification (supports) are managed using **SimLinkAccess** object:

- **AddLink (iName As String, iLink As AnyObject)**
- **RemoveAllLinks (iName As String)**
- **RemoveLink (iName As String, iLink As AnyObject)**
 - iName is the name of the link container, here it's "ConnectorList"

CATFmtLocalSpecLocalSize

This specification is used in order to specify a local mesh size or sag. Support, 0D, 1D, 2D or 3D supports are authorized

SizeValue

type: this argument is a <i>double</i> used in order to specify a local mesh size to <i>points, edges, faces or volumes of the geometry</i> .	Attribute which define a local size for mesh in a set of faces or edges.
--	--

AbsoluteSag

type: this argument is an <i>integer</i> used in order to specify a local mesh sag to <i>faces or edges of the geometry</i> :	<ul style="list-style-type: none"> • 1 if user doesn't want to apply a local mesh sag • 2 if user wants to Apply a local mesh sag <p>Attribute which specifies if the specification is created with an local sag on 1D or 2D supports. This attribute is linked to the AbsoluteSagValue attribute.</p>
--	--

AbsoluteSagValue

type: this argument is a length (<i>double</i>) value that specifies the local mesh sag value applied to <i>faces or edges of the geometry</i> .	<p>Specification which define a local sag for mesh in faces of edges. See also the definition of the sag.</p> <p>This attribute is used only when the AbsoluteSag attribute is active.</p>
---	--

CATFmtLocalMeshDistribution**NbElementsValue**

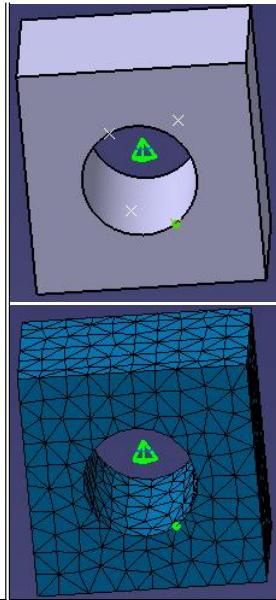
type: this attribute is an integer which specifies the number of mesh edges wanted on a specified geometry: Authorized supports: 1D and 2D supports are authorized only.	Specification which specifies an imposed mesh distribution on edges of the geometry.		
--	--	--	--

Size1Value

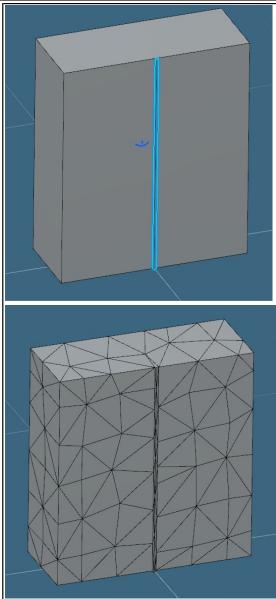
type: this attribute is a double value used in order to impose the size of mesh edges wanted on the specified edges of the geometry. Authorized supports: 1D and 2D supports are authorized only.	Specification which specifies an imposed mesh distribution edges size of the geometry.		
---	--	--	--

CATFmtImposedGeometries

--	--	--

<p>type: this specification is used in order to impose mesh (nodes) on points of the geometry.</p> <p>Authorized supports: 0D supports are authorized only.</p> <p>ToleranceValue: <i>double</i> valued to the maximum distance between external point and its projection on geometry to mesh.</p>	<p>Specification which specifies imposed nodes of the mesh.</p> <ul style="list-style-type: none"> if the distance between an external point and the geometry to mesh is greater than the ToleranceValue, the point will not be projected if the distance between an external point and the geometry to mesh is lower than the ToleranceValue, the point will be projected 	
--	--	---

CATFmtPreserveGeometries

<p>type: this specification is used in order to preserve mesh nodes and mesh edges of a specified mesh part geometry.</p> <p>Authorized supports: 0D, 1D and 2D supports are authorized only. For 2D support, faces boundaries are preserved</p>	<p>Specification which preserve nodes and edges of the mesh for a given geometry of mesh part.</p>	
---	--	--

History

Version: 1 [October 2014] Document created

Offset Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Offset Mesh Part** attributes and its local specifications.

[Offset Mesh Part Creation](#)

[Offset Mesh Part Support Definition](#)

[Global Mesh Specifications:](#)

- [Distance](#)
- [AutomaticMeshCapture](#)
- [AutomaticMeshCaptureTolValue](#)
- [NbCopies](#)

Offset Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Offset mesh part, the late type is "CATFmtOffsetMesher".

The Offset mesh part can be created by using **Add** method of **SimMeshParts** object.

Offset Mesh Part Support Definition

Offset Mesh Part support can be set using following method of the **SimMeshPart** object:

- **AddSupport (iSupport As AnyObject)** to add a new support to the mesh part

Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods of the **SimMeshPart** object:

- **SetAttributeValue (iSetType As String, iAttribute As String, iValue)** to set a mesh attribute where:
 - **iSetType**="Mesh"
 - **iAttribute** is the attribute's name
 - **iValue** is the attribute's value
- **GetAttributeValue (iSetType As String, iAttribute As String, oValue)** to get a mesh attribute where:
 - **iSetType**="Mesh"
 - **iAttribute** is the attribute's name
 - **oValue** is the attribute's value

Distance

type: <i>double</i> valuated to a length. default value: 0.2	Attribute which specifies the distance of the offset between the two meshes.
---	--

AutomaticMeshCapture

type: <i>integer</i> valuated to: • 1: to ignore automatic mesh capture • 2: to compute automatic mesh capture default value: 1	Attribute which specifies if external meshes must be captured. This attribute is linked to the attribute AutomaticMeshCaptureToValue .
--	---

AutomaticMeshCaptureToValue

type: <i>double</i> valuated to the maximum distance under which mesh is captured. default value: 0.	Attribute which specifies the maximum distance for mesh capture: if a mesh (element edges) is farthest from the support than this tolerance, it will not be captured even if the AutomaticMeshCapture attribute is valuated to 2. WARNING: to precise exactly the list of mesh parts to capture, the method SetMeshPartsToCapture of SimMeshPart object can be used: <ul style="list-style-type: none">• SetMeshPartsToCapture () To retrieve the parents mesh parts another method of SimMeshPart object can be used: <ul style="list-style-type: none">• Parent ()
---	--

NbCopies

type: <i>integer</i> valuated to the number of offsetted meshes. default value: 1	Attribute which specifies the number of offset meshes that we want to create.
--	---

History

Version: 1 [October 2014] Document created

Offset Extrusion Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Offset Extrusion Mesh Part** attributes and its local specifications.

[Offset Extrusion Mesh Part Creation](#)

[Offset Extrusion Mesh Part Support Definition](#)

[Global Mesh Specifications:](#)

- [Start](#)
- [End](#)
- [Type](#)
- [RatioValue](#)
- [NbElementsValue](#)

- [Symmetry](#)
- [AutomaticMeshCapture](#)
- [AutomaticMeshCaptureToValue](#)

Offset Extrusion Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Offset Extrusion mesh part, the late type is "CATFmtExtrusionOffsetMesher".

The Offset Extrusion mesh part can be created by using **Add** method of **SimMeshParts** object.

Offset Extrusion Mesh Part Support Definition

Offset Extrusion Mesh Part support can be set using following method of the **SimMeshPart** object:

- **AddSupport** (*iSupport As AnyObject*) to add a new support to the mesh part

Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods of the **SimMeshPart** object:

- **SetAttributeValue** (*iSetType As String, iAttribute As String, iValue*) to set a mesh attribute where:
 - **iSetType**="Mesh"
 - **iAttribute** is the attribute's name
 - **iValue** is the attribute's value
- **GetAttributeValue** (*iSetType As String, iAttribute As String, oValue*) to get a mesh attribute where:
 - **iSetType**="Mesh"
 - **iAttribute** is the attribute's name
 - **oValue** is the attribute's value

Start

type: <i>double</i> valued to a length. default value: 0.	Attribute which specifies the distance from the original mesh where the extrusion will begin. This attribute is linked to the attribute End .
--	--

End

type: <i>double</i> valued to a length. default value: 0.	Attribute which specifies the distance from the original mesh where the extrusion will stop. This attribute is linked to the attribute Start .
--	---

Type

type: <i>integer</i> valued to: <ul style="list-style-type: none"> • 1: to get an uniform distribution • 2: to get an arithmetic distribution • 3: to get a geometric distribution default value: 1	This attribute specifies the size distribution to be applied to the mesh. The picture shows how a uniform distribution (at the bottom) differs from a geometric one (at the top, using a ratio of 3).	
--	--	--

RatioValue

type: <i>double</i> valued to the size ratio between the first and last layer thicknesses. default value: 1.0	This attribute specifies the size ratio <i>R</i> between the first layer thickness <i>h1</i> and last layer thickness <i>h2</i> . $R = h2 / h1$ WARNING: It is taken into account only if the Type attribute is equal to 2 or 3.	
--	--	--

NbElementsValue

type: <i>integer</i> valued to the expected number of layers. default value: 1	This attribute specifies the number of layers to be added to the volume mesh.
---	---

Symmetry

type: boolean valued to create an extrusion which is symmetric.	Attribute which specifies if we want to create a symmetric extrusion, the symmetrical plane is located at the middle of the extrusion. WARNING: It is taken into account only if the Type attribute is equal to 2 or 3.	
--	--	--

AutomaticMeshCapture

type: integer valued to: <ul style="list-style-type: none">• 1: to ignore automatic mesh capture• 2: to compute automatic mesh capture default value: 1	Attribute which specifies if external meshes must be captured. This attribute is linked to the attribute AutomaticMeshCaptureTolValue .
--	--

AutomaticMeshCaptureTolValue

type: double valued to the maximum distance under which mesh is captured.	Attribute which specifies the maximum distance for mesh capture: if a mesh (element edges) is farthest from the support than this tolerance, it will not be captured even if the AutomaticMeshCapture attribute is valued to 2. WARNING: to precise exactly the list of mesh parts to capture, the method SetMeshPartsToCapture of SimMeshPart object can be used: <ul style="list-style-type: none">• SetMeshPartsToCapture () To retrieve the parents mesh parts another method of SimMeshPart object can be used: <ul style="list-style-type: none">• Parent ()
--	--

History

Version: 1 [November 2014] Document created

Partition Hex Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Partition Hex Mesh Part** attributes and its local specifications.

Partition Hex Mesh Part Creation**Partition Hex Mesh Part Support Definition****Global Mesh Specifications**

- [MeshSizeValue](#)
- [ElementOrder](#)
- [NonHexShapeAndOrder](#)

Local Mesh Specifications

- [CATFmtStudioDistributionSpec](#) (Edge Distribution):
 - [Type](#)
 - [Combination](#)
 - [CombinationUnif](#)
 - [NbElementsValue](#)
 - [RatioValue](#)
 - [Size1Value](#)
 - [Size2Value](#)
 - [Symmetry](#)

- o [Reverse](#)

Partition Hex Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the partition hex mesh part, the late type is "**CATFmtPartitionHexMesher**".

The partition hex mesh part can be created by using **Add** method of **SimMeshParts** object.

Partition Hex Mesh Part Support Definition

The Partition Hex Mesh Part support can be initialized and managed using the **AddSupport** method of **SimMeshPart** object:

- **AddSupport (iSupport As AnyObject)** to add a new support to the mesh part

Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods of **SimMeshPart** object:

- **SetAttributeValue (iSetType As String, iAttribute As String, iValue)** to set a mesh attribute
- **GetAttributeValue (iSetType As String, iAttribute As String, oValue)** to get a mesh attribute value

iAttribute is the name of specification and **iSetType = "Mesh"** for all Global Mesh Specifications.

MeshSizeValue

Type: **double**

valuated to the wanted size for mesh elements (it is an objective, Partition Hex mesher tried to approach an objective).

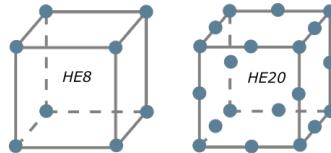
Attribute which specifies the global size of the mesh (it's an objective, Partition Hex mesher tried to approach nearest possible from this size).

ElementOrder

Type: **integer**

valuated to:

- 1: to get a mesh with Attribute which specifies the number of intermediate nodes linear hexaedron per element edge, as shown on the pictures (the left one (*HE8*) shows a linear hexahedron and the right one shows a quadratic hexahedron (*HE20*))
- 2: to get a mesh with parabolic hexahedron.



default value: 1

NonHexShapeAndOrder

Type: **integer**

valuated to:

- 1: to get a mesh with quadratic tetrahedrons (*TE10*).
- 2: to get a mesh with linear tetrahedrons (*TE4*).
- 3: to use a mesh with linear tetrahedrons and pyramids (*TE4 and PY5 or TE10 and PY13*)

This attribute specifies the which kind of elements will be used for the non hex elements.

default value: 1

Local Mesh Specifications

Local mesh specifications are created by the **Add** method of the **SimMeshSpecifications** object.

- **Add (iType As String)** with:
 - o **iType** is the late type of the mesh specification to create.

The **Item** and the **Remove** methods work with an index which is the position of the specification in the collection. The position index starts from 1.

The support of the local topology specification is managed by the **SimLinkAccess** object:

- **AddLink (iName As String, iLink As AnyObject)**
- **RemoveAllLinks (iName As String)**
- **RemoveLink (iName As String, iLink As AnyObject)**
 - o **iName** is the name of the link container, here it's "**ConnectorList**"

The specification values are managed by the following methods of **SimMeshSpecification**:

- **GetAttributeValue (iAttribute As String, oValue)**
- **SetAttributeValue (iAttribute As String, iValue)**
 - o **iAttribute** is the attribute name to set/retrieve.

CATFmtStudioDistributionSpec

Distribution specification (named for example *localSpec*) can be applied only on **one edge**

resulting of an unique grouping specification.
Specification attributes are managed using
SimMeshSpecification object:

- Geometric attribute (supports)

Other attributes are managed using
(GetAttributeValue, SetAttributeValue):

- **Type: integer** defining the type of distribution valued to:
 - 1: for a Uniform distribution (Isometric)
 - 2: for an Arithmetic distribution
 - 3: for a Geometric distribution
 - 4: for a distribution defined by a Law

default value: 1

- **Combination: integer** indicating what are the possible combination for Arithmetic or Geometric distributions valued to:
 - 1: for a distribution defined by the number of elements (*NbElements*) and the size ratio of the elements (*Ratio21*)
 - 2: for a distribution defined by the number of elements (*NbElements*) and the size (*Size1*) at first vertex
 - 3: for a distribution defined by the number of elements (*NbElements*) and the size (*Size2*) at second vertex
 - 4: for a distribution defined by the size ratio (*Ratio21*) and the size (*Size1*) at first vertex
 - 5: for a distribution defined by the size ratio (*Ratio21*) and the size (*Size2*) at second vertex
 - 6: for a distribution defined by the size (*Size1*) at first vertex and the size (*Size2*) at second vertex

default value: 1

- **CombinationUnif: integer** indicating what are the possible combination for Uniform distributions valued to:
 - 1: for a distribution defined by the number of elements (*NbElements*)
 - 2: for a distribution defined by the size (*Size1*) of the elements

default value: 1

- **NbElementsValue: integer** defining the number of elements (*NbElements*)

default value: 4

- **RatioValue: double** defining the ratio between the max element size and the min element size (*Ratio21*)

default value: 1.

- **Size1Value: double** defining the size at first vertex (*Size1*)

default value: 0.

- **Size2Value: double** defining the size at second vertex (*Size2*)

default value: 0.

- **Symmetry: integer** valued to:
 - 1: to keep the distribution unchanged
 - 2: to compute a symmetric distribution with current attributes

default value: 1

- **Reverse: integer** usable for Law and non Uniform distribution valued to:
 - 1: to keep the distribution unchanged
 - 2: to invert the distribution

default value: 1

Rotation Mesh Part Attributes

Technical Article

Description: This specification creates a node distribution on an edge resulting of a 1D grouping specification. Depending on the choice of the distribution type (Type attribute), Combination or CombinationUnif attributes several various distributions "Option" can be computed. The next table summarize all "Option":

- Isometric (Uniform) distribution is defined by only one attributes(Option 1 and 2):
 - *Nbelements* (**CombinationUnif= 1**)
 - *Size1* (**CombinationUnif= 2**)
- Non isometric distribution is defined by two attributes as indicated in the next table. There are six couples of attributes (Option 3 to 8):
 - *Nbelements* and *Ratio21* (**Combination = 1**)
 - *NbElements* and *Size1* (**Combination = 2**)
 - *NbElements* and *Size2* (**Combination = 3**)
 - *Ratio21* and *Size1* (**Combination = 4**)
 - *Ratio21* and *Size2* (**Combination = 5**)
 - *Size1* and *Size2* (**Combination = 6**)

	"Isometric"		"Arithmetic"\ "Geometric"					
	1	2	3	4	5	6	7	8
"Option"	●		●	●	●			
"NbElements"	●							
"Ratio21"			●			●	●	
"Size1"		●		●		●		●
"Size2"					●	●	●	●

The next table shows examples of various distributions using additional Symmetry and Law attributes.

Attributes	Examples				
	Type	«Arithmetic»	«Geometric»	«Uniform»	«Law»
NbElements	●	10	●	●	●
Symmetric	●	Yes	●	●	●
Size1	50	●	15	●	●
Size2	200	●	●	●	●
Ratio21	●	2	●	●	●
Reverse	●	●	●	●	●
Law	●	●	●	●	$y=x^2$

Abstract

This article presents an overview of the **Rotation Mesh Part** attributes and its local specifications.

Rotation Mesh Part Creation

RotationMesh Part Support Definition:

- [Geometry](#)

Global Mesh Specifications:

- [Angle](#)
- [AutomaticMeshCapture](#)
- [AutomaticMeshCaptureTolValue](#)
- [NbCopies](#)

Rotation Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Rotation mesh part, the late type is "CATFmtRotationMesher".

The Rotation mesh part can be created by using **Add** method of **SimMeshParts** collection object.

Rotation Mesh Part Support Definition

Rotation Mesh Part support can be set using following method of the **SimMeshPart** object:

- **AddSupport** (*iSupport As AnyObject*) to add a new support to the mesh part

Geometry

The Rotation Mesh Part axis can be set or unset using following methods of the **SimMeshPart** object:

- **SetExternalReference** (*iSetType As String*, *iAttribute As String*, *iSupport As AnyObject*) to set the rotation axis where:
 - *iSetType*="Topology"
 - *iAttribute*="Geometry"
 - *iSupport* is a geometric line
- **RemoveExternalReference** (*iSetType As String*, *iAttribute As String*, *iSupport As AnyObject*) to unset the rotation axis where:
 - *iSetType*="Topology"
 - *iAttribute*="Geometry"
 - *iSupport* is a geometric line

Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods of the **SimMeshPart** object:

- **SetAttributeValue** (*iSetType As String*, *iAttribute As String*, *iValue*) to set a mesh attribute where:
 - *iSetType*="Mesh"
 - *iAttribute* is the attribute's name
 - *iValue* is the attribute's value
- **GetAttributeValue** (*iSetType As String*, *iAttribute As String*, *oValue*) to get a mesh attribute where:
 - *iSetType*="Mesh"
 - *iAttribute* is the attribute's name
 - *oValue* is the attribute's value

Angle

type: <i>double</i> valued to an angle.	Attribute which specifies the angle between the original mesh and the new mesh that we want to create thanks to a Rotation transformation.
default value: 90deg	

AutomaticMeshCapture

type: <i>integer</i> valued to:	Attribute which specifies if external meshes must be captured. This attribute is linked to the attribute AutomaticMeshCaptureTolValue .
<ul style="list-style-type: none"> • 1: to ignore automatic mesh capture • 2: to compute automatic mesh capture default value: 1	

AutomaticMeshCaptureTolValue

type: <i>double</i> valued to the maximum distance under which mesh is captured.	Attribute which specifies the maximum distance for mesh capture: if a mesh (element edges) is farthest from the support than this tolerance, it will not be captured even if the AutomaticMeshCapture attribute is valued to 2.
default value: 0.	<p>WARNING: to precise exactly the list of mesh parts to capture, the method SetMeshPartsToCapture of SimMeshPart object can be used:</p> <ul style="list-style-type: none"> • SetMeshPartsToCapture ()

To retrieve the parents mesh parts another method of **SimMeshPart** object can be used:

- Parent ()

NbCopies

type: <i>integer</i> valued to the number of translated meshes. default value: 1	Attribute which specifies the number of translated meshes that we want to create.
---	---

History

Version: 1 [October 2014] Document created

Rotation Extrusion Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Rotation Extrusion Mesh Part** attributes and its local specifications.

[Rotation Extrusion Mesh Part Creation](#)

[Rotation Extrusion Mesh Part Support Definition:](#)

- [Geometry](#)

[Global Mesh Specifications:](#)

- [Start](#)
- [End](#)
- [Type](#)
- [RatioValue](#)
- [NbElementsValue](#)
- [Symmetry](#)
- [AutomaticMeshCapture](#)
- [AutomaticMeshCaptureTolValue](#)

Rotation Extrusion Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Rotation Extrusion mesh part, the late type is "CATFmtExtrusionRotationMesher".

The Rotation Extrusion mesh part can be created by using **Add** method of **SimMeshParts** object.

Rotation Extrusion Mesh Part Support Definition

Rotation Extrusion Mesh Part support can be set using following method of the **SimMeshPart** object:

- **AddSupport** (*iSupport As AnyObject*) to add a new support to the mesh part

Geometry

The Rotation Extrusion Mesh Part axis can be set or unset using following methods of the **SimMeshPart** object:

- **SetExternalReference** (*iSetType As String, iAttribute As String, iSupport As AnyObject*) to set the rotation axis where:
 - *iSetType*="Topology"
 - *iAttribute*="Geometry"
 - *iSupport* is a geometric line
- **RemoveExternalReference** (*iSetType As String, iAttribute As String, iSupport As AnyObject*) to unset the rotation axis where:
 - *iSetType*="Topology"
 - *iAttribute*="Geometry"
 - *iSupport* is a geometric line

Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods of the **SimMeshPart** object:

- **SetAttributeValue** (*iSetType As String, iAttribute As String, iValue*) to set a mesh attribute where:
 - *iSetType*="Mesh"
 - *iAttribute* is the attribute's name
 - *iValue* is the attribute's value
- **GetAttributeValue** (*iSetType As String, iAttribute As String, oValue*) to get a mesh attribute where:
 - *iSetType*="Mesh"
 - *iAttribute* is the attribute's name
 - *oValue* is the attribute's value

Start

type: *double* valued to an angle.

default value: 0.deg

Attribute which specifies the angle from the original mesh where the extrusion will begin. This attribute is linked to the attribute **End**.

End

type: *double* valued to an angle.

default value: 0.deg

Attribute which specifies the angle from the original mesh where the extrusion will stop. This attribute is linked to the attribute **Start**.

Type

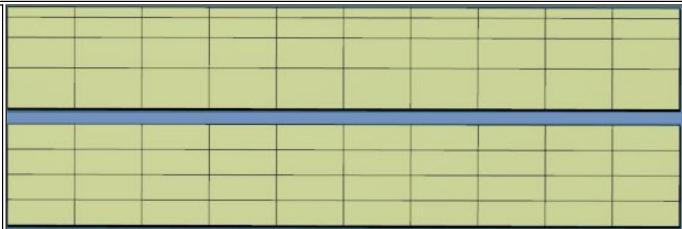
type: *integer* valued to:

- 1: to get an uniform distribution
- 2: to get an arithmetic distribution
- 3: to get a geometric distribution

default value: 1

This attribute specifies the size distribution to be applied to the mesh.

The picture shows how a uniform distribution (at the bottom) differs from a geometric one (at the top, using a [ratio](#) of 3).



RatioValue

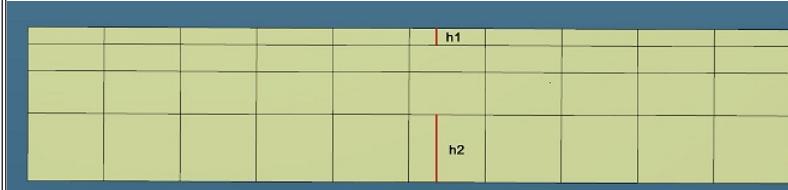
type: *double* valued to the size ratio between the first and last layer thicknesses.

default value: 1.0

This attribute specifies the size ratio R between the first layer thickness $h1$ and last layer thickness $h2$.

$$R = h2 / h1$$

WARNING: It is taken into account only if the **Type** attribute is equal to 2 or 3.



NbElementsValue

type: *integer* valued to the expected number of layers.

default value: 1

This attribute specifies the number of layers to be added to the volume mesh.

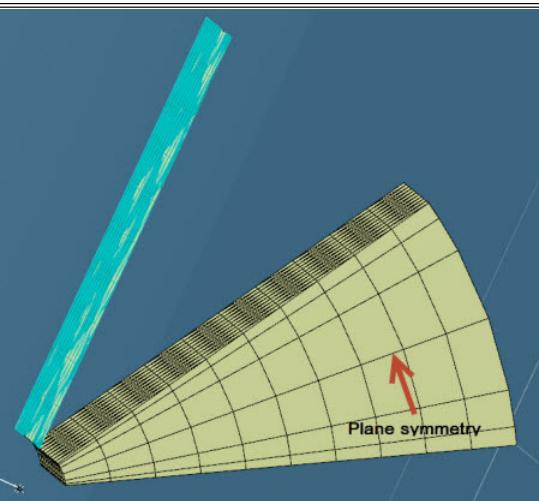
Symmetry

type: *boolean* valued to create an extrusion which is symmetric.

default value: false

Attribute which specifies if we want to create a symmetrical extrusion, the symmetrical plane is located ad the middle of the extrusion.

WARNING: It is taken into account only if the **Type** attribute is equal to 2 or 3.



AutomaticMeshCapture

type: *integer* valued to:

- 1: to ignore automatic mesh capture
- 2: to compute automatic mesh capture

default value: 1

Attribute which specifies if external meshes must be captured. This attribute is linked to the attribute **AutomaticMeshCaptureToValue**.

AutomaticMeshCaptureTolValue

<p>type: <code>double</code> valuated to the maximum distance under which mesh is captured.</p> <p>default value: 0.</p>	<p>Attribute which specifies the maximum distance for mesh capture: if a mesh (element edges) is farthest from the support than this tolerance, it will not be captured even if the AutomaticMeshCapture attribute is valuated to 2.</p> <p>WARNING: to precise exactly the list of mesh parts to capture, the method SetMeshPartsToCapture of SimMeshPart object can be used:</p> <ul style="list-style-type: none"> • <code>SetMeshPartsToCapture ()</code> <p>To retrieve the parents mesh parts another method of SimMeshPart object can be used:</p> <ul style="list-style-type: none"> • <code>Parent ()</code>
--	--

History

Version: 1 [November 2014] Document created

Spine Extrusion Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Spine Extrusion Mesh Part** attributes and its local specifications.

Spine Extrusion Mesh Part Creation**Spine Extrusion Mesh Part Support Definition:**

- [Geometry](#)

Global Mesh Specifications:

- [Start](#)
- [End](#)
- [AnchorPoint](#)
- [Type](#)
- [RatioValue](#)
- [NbElementsValue](#)
- [Symmetry](#)
- [AutomaticMeshCapture](#)
- [AutomaticMeshCaptureTolValue](#)

Spine Extrusion Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Spine Extrusion mesh part, the late type is "CATFmtExtrusionSpineMesher".

The Spine Extrusion mesh part can be created by using **Add** method of **SimMeshParts** object.

Spine Extrusion Mesh Part Support Definition

Spine Extrusion Mesh Part support can be set using following method of the **SimMeshPart** object:

- **AddSupport** (*iSupport As AnyObject*) to add a new support to the mesh part

Geometry

The Spine Extrusion Mesh Part spine can be set or unset using following methods of the **SimMeshPart** object:

- **SetExternalReference** (*iSetType As String, iAttribute As String, iSupport As AnyObject*) to set the spine where:
 - *iSetType*="Topology"
 - *iAttribute*="Geometry"
 - *iSupport* is a geometric spine
- **RemoveExternalReference** (*iSetType As String, iAttribute As String, iSupport As AnyObject*) to unset the spine where:
 - *iSetType*="Topology"
 - *iAttribute*="Geometry"
 - *iSupport* is a geometric spine

Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods of the **SimMeshPart** object:

- **SetAttributeValue** (*iSetType As String, iAttribute As String, iValue*) to set a mesh attribute where:
 - *iSetType*="Mesh"
 - *iAttribute* is the attribute's name
 - *iValue* is the attribute's value
- **GetAttributeValue** (*iSetType As String, iAttribute As String, oValue*) to get a mesh attribute where:
 - *iSetType*="Mesh"
 - *iAttribute* is the attribute's name
 - *oValue* is the attribute's value

Start

type: <i>double</i> valued to a length. default value: 0.	Attribute which specifies the distance from the original mesh where the extrusion will begin. This attribute is linked to the attribute End .
--	--

End

type: <i>double</i> valued to a length. default value: 0.	Attribute which specifies the distance from the original mesh where the extrusion will stop. This attribute is linked to the attribute Start .
--	---

AnchorPoint

type: <i>point</i> valued to define the beginning of the curve that we take into account for the extrusion	Attribute which specifies the first point from the spine curve that the extrusion algorithm will take into account in order to generate the extruded mesh.
---	--

Type

type: <i>integer</i> valued to: <ul style="list-style-type: none">• 1: to get an uniform distribution• 2: to get an arithmetic distribution• 3: to get a geometric distribution default value: 1	This attribute specifies the size distribution to be applied to the mesh. The picture shows how a uniform distribution (at the bottom) differs from a geometric one (at the top, using a ratio of 3).	
--	---	--

RatioValue

type: <i>double</i> valued to the size ratio between the first and last layer thicknesses. default value: 1.0	This attribute specifies the size ratio R between the first layer thickness $h1$ and last layer thickness $h2$. $R = h2 / h1$ WARNING: It is taken into account only if the Type attribute is equal to 2 or 3.	
--	---	--

NbElementsValue

type: <i>integer</i> valued to the expected number of layers. default value: 1	This attribute specifies the number of layers to be added to the volume mesh.
---	---

Symmetry

type: <i>boolean</i> valued to create an extrusion which is symmetric. default value: false	Attribute which specifies if we want to create a symmetric extrusion, the symmetrical plane is located at the middle of the extrusion. WARNING: It is taken into account only if the Type attribute is equal to 2 or 3.	
--	--	--

AutomaticMeshCapture

type: <i>integer</i> valued to:	
<ul style="list-style-type: none"> • 1: to ignore automatic mesh capture • 2: to compute automatic mesh capture 	Attribute which specifies if external meshes must be captured. This attribute is linked to the attribute AutomaticMeshCaptureTolValue .
default value: 1	

AutomaticMeshCaptureTolValue

type: <i>double</i> valued to the maximum distance under which mesh is captured.	Attribute which specifies the maximum distance for mesh capture: if a mesh (element edges) is farthest from the support than this tolerance, it will not be captured even if the AutomaticMeshCapture attribute is valued to 2.
default value: 0.	<p>WARNING: to precise exactly the list of mesh parts to capture, the method SetMeshPartsToCapture of SimMeshPart object can be used:</p> <ul style="list-style-type: none"> • SetMeshPartsToCapture () <p>To retrieve the parents mesh parts another method of SimMeshPart object can be used:</p> <ul style="list-style-type: none"> • Parent ()

History

Version: 1 [November 2014] Document created

Surface Quadrangle Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Surface Quadrangle Mesh Part** attributes and its local specifications.

Surface Quadrangle Mesh Part Creation:**Surface Quadrangle Mesh Part Support Definition:**

- [CleanedGeometry](#)
- [CleaningMode](#)

Global Mesh Specifications:

- [MeshSizeValue](#)
- [ElementShape](#)
- [AutoMap](#)
- [ElementOrder](#)
- [AutomaticMeshCapture](#)
- [AutomaticMeshCaptureTolValue](#)

Global Topology Specifications:

- [AngleBetweenCurves](#)
- [AngleBetweenFaces](#)
- [CurvatureAngle](#)
- [OffsetValue](#)
- [OffsetFromThickness](#)
- [AutomaticCurveCapture](#)
- [GeometrySimplification](#)
- [GeometrySimplificationValue](#)
- [2DHOlesSuppression](#)
- [2DHOlesMaxDiameter](#)
- [LogosSuppression](#)
- [LogosMaxHeight](#)
- [LogosMaxSize](#)

Local Specifications:**Local Topology Specifications:**

- [CATFmtConstrainEdgeLocalSpec](#) (Constrain Edge)
- [CATFmtUnconstrainEdgeLocalSpec](#) (Unconstrain Edge)
- [CATFmtConstrainHoleLocalSpec](#) (Constrain Hole)
- [CATFmtUnconstrainHoleLocalSpec](#) (Unconstrain Hole)
- [CATFmtGroupingLocalSpec](#) (Group Geometries)
- [CATFmtExternalPointLocalSpec](#) (Imposed Point):
 - [ToleranceValue](#)
 - [ProjectOnGeometry](#)
- [CATFmtExternalCurveLocalSpec](#) (Imposed Curve):
 - [ToleranceValue](#)
- [CATFmtExternalCurveWeldLocalSpec](#) (Imposed Weld Curve):
 - [ReuseConnectionParameters](#)
 - [ToleranceValue](#)
 - [NbRows](#)
 - [Height1](#)
 - [Height2](#)
 - [Height3](#)

- [SelfWelding](#)

Local Mesh 1D Specifications:

- [CATFmtStudioDistributionSpec](#) (Edge Distribution):
 - [Type](#)
 - [Combination](#)
 - [CombinationUnif](#)
 - [NbElementsValue](#)
 - [RatioValue](#)
 - [Size1Value](#)
 - [Size2Value](#)
 - [Symmetry](#)
 - [Reverse](#)
- [CATFmtStudioHoleSpec](#) (Edge Distribution around Hole):
 - [NbElements](#)
 - [NbRows](#)
 - [Heighth1](#)
 - [Heighth2](#)
 - [Heighth3](#)
- [CATFmtStudioBoundaryLayerSpec](#) (Edges Distribution around Curves):
 - [NbRows](#)
 - [Heighth1](#)
 - [Heighth2](#)
 - [Heighth3](#)
- [CATFmtStudioCaptureSpec](#) (Element Capture):
 - [ManualMeshCaptureMode](#)
 - [ManualMeshCaptureTolValue](#)
- [CATFmtStudioPropagationSpec](#) (Distribution Propagation):
 - [PropagationMode](#)
 - [Geometry](#)

Local Mesh 2D Specifications:

- [CATFmtStudioFrontQuadSpec](#) (Frontal Quadrangle Method):
 - [SizeValue](#)
- [CATFmtStudioFrontTriaSpec](#) (Frontal Triangle Method):
 - [SizeValue](#)
- [CATFmtStudioMappedSpec](#) (Mapped Method):
 - [SizeValue](#)
- [CATFmtStudioMappedFreeMinimalSpec](#) (Mapped Minimal Method):
 - [TriangularShape](#)
- [CATFmtStudioMappedFreeSpec](#) (Mapped Free Method):
 - [SizeValue](#)
 - [TriangularShape](#)
- [CATFmtStudioBeadSpec](#) (Bead Method):
 - [SizeValue](#)
- [CATFmtStudioHalfBeadSpec](#) (Half Bead Method):
 - [SizeValue](#)
- [CATFmtStudioFaceCaptureSpec](#) (Face Capture):
 - [MeshSources](#)
 - [Tolerance](#)
 - [ManualMeshCaptureMode](#)

Surface Quadrangle Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Surface Quadrangle Mesh Part, the late type is "[CATFmtSurfQuadMesher](#)".

The Surface Quadrangle Mesh Part can be created by using **Add** method of **SimMeshParts** object.

Surface Mesh Part Support Definition

The Surface Quadrangle Mesh Part support can be initialized and managed using the **AddSupport** method of **SimMeshPart** object:

- **AddSupport** (*iSupport As AnyObject*) to add a new support to the mesh part

CleanedGeometry

Surface Quadrangle Mesh Part support can be optionally modified to keep a subset of the previous geometry. For that purpose, the list of geometry to keep or to remove has to be managed using the following methods with **iSetType = "Topology"** and **iAttribute = "CleanedGeometry"**:

- **SetExternalReference** (*iSetType As String*, *iAttribute As String*, *iSupport As AnyObject*) to add external reference to the mesh part
- **RemoveExternalReference** (*iSetType As String*, *iAttribute As String*, *iSupport As AnyObject*) to remove external reference from the mesh part

Other mesh part attributes support definition are managed using following methods:

- **SetAttributeValue** (*iSetType As String*, *iAttribute As String*, *iValue*) to set a mesh attribute with an integer value
- **GetAttributeValue** (*iSetType As String*, *iAttribute As String*, *oValue*) to get a mesh attribute with an integer value

where **iType** is the type of specification and **iType = "Topology"** for support definition and **iName** the attribute name.

CleaningMode

type: <i>integer</i> valued to:	1: to exclude geometry of CleanedGeometry of support	Attribute which specifies if the geometries stored in CleanedGeometry external references have to
--	---	---

- | | |
|---|--|
| <ul style="list-style-type: none"> • 2: to keep only geometry of CleanedGeometry as support <p>default value: 1</p> | be removed from the mesh part support definition or if this geometries will replace the initial support. |
|---|--|

Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods:

- **SetAttributeValue** (*iSetType As String, iAttribute As String, iValue*) to set a mesh attribute with an integer value
- **GetAttributeValue** (*iSetType As String, iAttribute As String, oValue*) to get a mesh attribute with an integer value

iType is the type of specification and **iType = "Mesh"** for Global Mesh Specifications.

MeshSizeValue

type: <i>double</i> valued to mesh elements size default value: 0.0	Attribute which specifies the mesh element target size that surface mesher tries to respect.
--	--

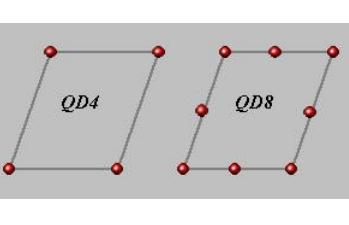
ElementShape

type: <i>integer</i> valued to: <ul style="list-style-type: none"> • 1: to get a mixed mesh with triangle and quadrangle elements • 2: to get a mesh only with quadrangle elements • 3: to get a mesh with a minimal number of quadrangle and triangle elements default value: 1	<p>There are two different elements types: quadrangle elements and triangle elements.</p> <p>Attribute which specifies the type of mesh. The mesh can have:</p> <ul style="list-style-type: none"> • both triangle and quadrangle elements with usually more quadrangle than triangle elements ("Quads Dominant") • only quadrangle elements ("Quads Only") • both triangle and quadrangle elements ("Quads Minimum"). Only topologically and geometrically (4 corners) valid domains are taken into account.
--	--

AutoMap

type: <i>integer</i> valued to: <ul style="list-style-type: none"> • 1: do not map mesh the support • 2: map mesh the support where appropriate default value: 1	<p>This attribute specifies whether map mesh is applied on support where appropriate.</p> <p>WARNING: It is taken into account only if the ElementShape attribute is equal to 1.</p>
---	--

ElementOrder

type: <i>integer</i> valued to: <ul style="list-style-type: none"> • 1: to get a mesh with linear elements (<i>TR3</i> or <i>QD4</i> depending on Element Shape choice) • 2: to get a mesh with quadratic elements (<i>TR6</i> or <i>QD8</i> depending on Element Shape choice) default value: 1	<p>Attribute which specifies the number of intermediate nodes per element edge, for example in a case of quadrangle element we get linear QD4 (on the left picture) or quadratic QD8 (on the right picture).</p>	
---	---	---

AutomaticMeshCapture

type: <i>integer</i> valued to: <ul style="list-style-type: none"> • 1: to ignore automatic mesh capture • 2: to compute automatic mesh capture default value: 1	<p>Attribute which specifies if external meshes must be captured. This attribute is linked to the attribute AutomaticMeshCaptureTolValue.</p>
---	--

AutomaticMeshCaptureTolValue

type: <i>double</i> valued to the maximum distance under which mesh is captured. default value: 0.	<p>Attribute which specifies the maximum distance for mesh capture: if a mesh (element edges) is farthest from the geometry than this tolerance, it will not be captured even if the AutomaticMeshCapture attribute is valued to 2.</p> <p>WARNING: to precise exactly the list of mesh parts to capture, the method SetMeshPartsToCapture of SimMeshPart interface can be used:</p> <ul style="list-style-type: none"> • SetMeshPartsToCapture () <p>To retrieve the parents mesh parts another method of SimMeshPart interface can be used:</p>
---	--

• Parent

Global Topology Specifications

Global topology specifications can be set or retrieved using following methods:

- **SetAttributeValue** (*iSetType As String, iAttribute As String, iValue*) to set a topology attribute
- **GetAttributeValue** (*iSetType As String, iAttribute As String, oValue*) to get a topology attribute

iType is the type of specification and **iType** = "Topology" for Global Topology Specifications.

AngleBetweenCurves

<p>type: double (radian unit) valuated to the angle under which vertices are inactivated default value: 20°</p>	<p>Attribute which specifies the maximum angle under which a vertex connected exactly to two constrained edges can be inactivated.</p>	
---	--	--

AngleBetweenFaces

<p>type: double (radian unit) valuated to the angle under which edges are inactivated. default value: 20°</p>	<p>Attribute which specifies the maximum angle under which an edge connected exactly to two faces can be inactivated during topology computation.</p>	
---	---	--

CurvatureAngle

<p>type: double (radian unit) valuated to the angle over which edges are activated. default value: 20°</p>	<p>Attribute which specifies the minimum angle computed along filleted surfaces over which edges are activated. For a given fillet, when the computed curvature angle is greater than CurvatureAngle all boundary edges of this fillet are activated otherwise they are not.</p>	
--	---	--

OffsetValue

<p>type: double valuated to the offset between the geometry and the mesh. default value: 0.</p>	<p>Attribute which specifies the distance of the offset between the mesh and the geometry.</p>	
---	--	--

OffsetFromThickness

<p>type: integer valued to:</p> <ul style="list-style-type: none"> • 1: if OffsetValue has not been computed from geometry thickness • 2: if OffsetValue has been computed from geometry thickness <p>default value: 0.</p>	<p>Attribute which specifies if OffsetValue has been computed from geometry thickness using for example CATIGSMThicknessinterface.</p>
---	---

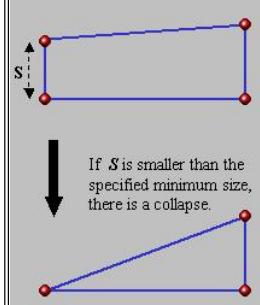
AutomaticCurveCapture

<p>type: integer valued to:</p> <ul style="list-style-type: none"> • 1: to ignore automatic capture of external curves • 2: to activate automatic capture of external curves <p>default value: 1</p>	<p>Attribute which specifies if external curves must be captured automatically. It is possible to automatically capture curve only when AutomaticMeshCapture attribute value is set to 2 (enabled). The geometries used to compute the projections are the support of the mesh part to be captured. The tolerance used to compute the projections is AutomaticMeshCaptureTolValue.</p>
--	--

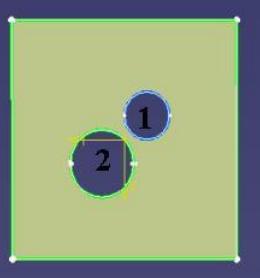
GeometrySimplification

type: <i>integer</i> valued to: • 1: to keep grouping topology simplification without additional simplification • 2: to compute topology simplification default value: 2	Attribute which specifies if topology simplification is computed. Topology simplification objective is to minimize the number of bad elements regarding the Min Height element quality criterion.
---	---

GeometrySimplificationValue

type: <i>double</i> valued to the Min Height criterion targeted for mesh elements. default value: 0.5 * MeshSizeValue	When geometry simplification is computed, various topology modifications are done automatically using a set of topological operators in order to respect GeometrySimplificationValue (kind of minimum element size quality criterion): • Vertex Creation/Activation/Inactivation • Edge Creation/Activation/Inactivation • Edge collapse (see picture) • Edge Merge	
--	--	---

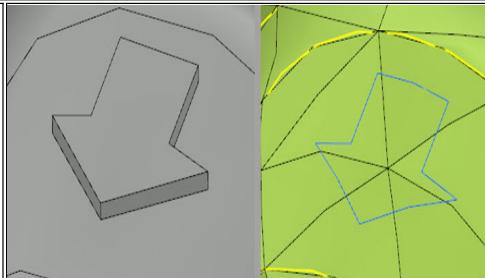
2D Holes Suppression

type: <i>integer</i> valued to: • 1: to ignore hole control suppression • 2: to control hole suppression default value: 1	Attribute which specifies whether hole suppression will be done. Hole suppression is computed from hole diameter value. For example in the right picture, a suppress hole size (2D Holes Max Diameter attribute) was specified for the mesh part. Hole 1 diameter is below than this value and was ignored conversely to Hole 2.	
--	---	--

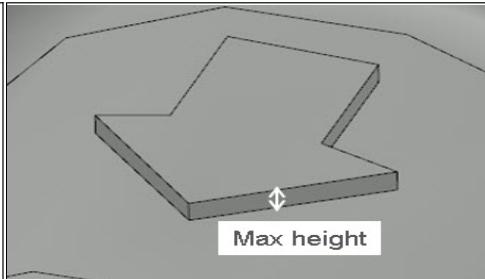
2D Holes Max Diameter

type: <i>double</i> valued to the maximum diameter under which a hole is ignored. default value: 0.	Attribute which specifies the maximum diameter allowed for hole suppression. This value is computed as the maximum distance between two points of a hole discretization. WARNING: It is taken into account only if the 2D Holes Suppression attribute is equal to 2.
--	---

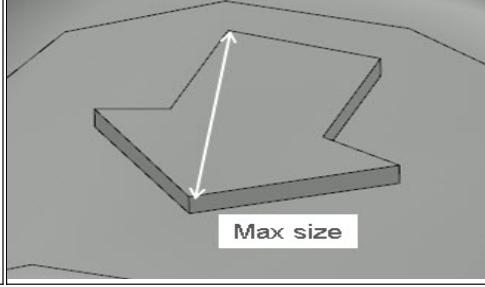
Logos Suppression

type: <i>integer</i> valued to: • 1: to ignore logos control suppression • 2: to control logos suppression default value: 1	Attribute which specifies whether logos suppression will be done. Logos suppression consists of ignoring small features of the geometry that are useless for the mesh (small height regarding the mesh size). Logos suppression is driven by two attributes: LogosMaxHeight and LogosMaxSize . For example on the right picture, an ignored logo can be seen.	
--	---	--

LogosMaxHeight

type: <i>double</i> valued to the max height of a detectable logo. default value: 0.1 * MeshSizeValue	Attribute which specifies the maximal height allowed to detect logos. The maximum height of a logo can be defined as the maximum distance of any points on the logo faces to the extrapolated bounding surfaces.	
--	---	--

LogosMaxSize

<p>type: <i>double</i> valued to the max size of a detectable logo</p> <p>default value: <i>MeshSizeValue</i></p>	<p>Attribute which specifies the maximal size allowed to detect logos.</p> <p>The maximum size of a logo can be defined as the maximum distance between two points on the logo boundary (maximum diameter).</p>	
---	---	--

Local Specifications

The Surface Quadrangle Mesh Part has two kind of local specifications: **topology** and **mesh** local specifications. These specifications are managed by collection object:

- **SimMeshSpecifications:** for the collection of local mesh specifications
- **SimTopologySpecifications:** for the collection of local topology specifications

You should use the following methods of **SimMeshPart** object to retrieve these collections:

- **GetMeshSpecifications** to retrieve the local mesh specification collection
- **GetTopologySpecification** to retrieve the local topology specification collection

Local specifications are managed from the collection objects:

- **Add** (*iType As String*) to create a new local specification with:
 - *iType* is the late type of the specification to create
- **Item** (*iIndex*) to retrieve a local specification with:
 - *iIndex* is the index of the local specification in its collection
- **Remove** (*iIndex*) to delete a local specification with:
 - *iIndex* is the index of the local specification in its collection

Attribute of local specification are set using following method from **SimMeshSpecification** or **SimTopologySpecification** object depending of the kind of local specification:

- **SetAttributeValue** (*iAttribute As String, iValue*) to set a local specification attribute value

In a manner similar, attribute of local specification are retrieved using following method:

- **GetAttributeValue** (*iAttribute As String, iValue*) to retrieve a local specification

Geometric attribute of local specification (supports) are managed using **SimLinkAccess** object:

- **AddLink** (*iName As String, iLink As AnyObject*)
- **RemoveAllLinks** (*iName As String*)
- **RemoveLink** (*iName As String, iLink As AnyObject*)
 - *iName* is the name of the link container, here it's "**ConnectorList**"

Local Topology Specifications

CATFmtConstrainEdgeLocalSpec

<p>Specification attributes are managed using SimTopologySpecification object:</p> <ul style="list-style-type: none"> • Only geometric attribute (supports) 	<p>Description: This specification is used in order to constrain a set of edges.</p>
---	---

CATFmtUnconstrainEdgeLocalSpec

<p>Specification attributes are managed using SimTopologySpecification object.</p> <ul style="list-style-type: none"> • Only geometric attribute (supports) 	<p>Description: This specification is used in order to unconstrain a set of edges.</p>
---	---

CATFmtConstrainHoleLocalSpec

<p>Specification attributes are managed using SimTopologySpecification object.</p> <ul style="list-style-type: none"> • Only geometric attribute (supports) 	<p>Description: This specification is used in order to constrain a hole.</p>
---	---

CATFmtUnconstrainHoleLocalSpec

<p>Specification attributes are managed using SimTopologySpecification object.</p> <ul style="list-style-type: none"> • Only geometric attribute (supports) 	<p>Description: This specification is used in order to unconstrain a hole.</p>
---	---

CATFmtGroupingLocalSpec

	<p>Description: This specification is used in order to group geometries from a set of supports (internal 1D or 2D geometries that belong to the geometry to mesh). The overall behavior is the</p>
--	---

<p>Specification attributes are managed using SimTopologySpecification object.</p> <ul style="list-style-type: none"> • Only geometric attribute (supports) 	<p>following:</p> <ul style="list-style-type: none"> • if 1D geometries have been selected the program tries to create an edge of the topology per group of connected selected 1D geometries. 1D selected geometries are also constrained during this execution. <p>WARNING: It is possible to select only one support which useful to set local mesh specifications</p> <ul style="list-style-type: none"> • if 2D geometries have been selected the program tries to create a domain of the topology per group of connected selected 2D geometries. Consequently, all boundary edges of new created domains are constrained and all internal edges of new created domains are unconstrained.
---	--

CATFmtExternalPointLocalSpec

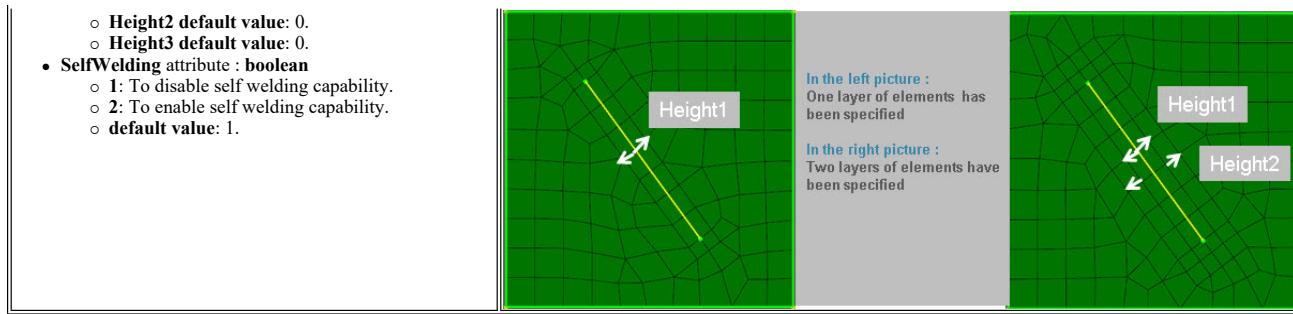
<p>Specification attributes are managed using SimTopologySpecification interface:</p> <ul style="list-style-type: none"> • Geometric attribute (supports) <p>Other attributes are managed using (GetAttributeValue, SetAttributeValue):</p> <ul style="list-style-type: none"> • ToleranceValue: double valued to the maximum distance between external point and its projection on geometry to mesh. <p>default value: 0.</p> <ul style="list-style-type: none"> • ProjectOnGeometry: integer valued to the target geometry choice <p>default value: 1</p>	<p>Description: This specification is used in order to project external points (points that don't belong to the geometry to mesh) in order to constrain the mesh. External points definition (supports) and projection tolerance are mandatory. The overall behavior is the following:</p> <ul style="list-style-type: none"> • if the distance between an external point and the geometry to mesh is greater than the ToleranceValue, the point will not be projected • if the distance between an external point and the geometry to mesh is lower than the ToleranceValue, the point will be projected. The final location of this point depends on ProjectOnGeometry attribute: <ul style="list-style-type: none"> ◦ if ProjectOnGeometry is equal to 1, the node representing the point will be located exactly on the projection point on the surface geometry ◦ if ProjectOnGeometry is equal to 2, the node representing the point projection will be located exactly on the external point (same Cartesian coordinates)
---	--

CATFmtExternalCurveLocalSpec

<p>Specification attributes are managed using SimTopologySpecification interface:</p> <ul style="list-style-type: none"> • Geometric attribute (supports) <p>Other attributes are managed using (GetAttributeValue, SetAttributeValue):</p> <ul style="list-style-type: none"> • ToleranceValue attribute is set and retrieved with: <ul style="list-style-type: none"> ◦ SetAttributeValue ◦ GetAttributeValue <p>type: double valued to the maximum distance between external curves and its projection on geometry to mesh.</p> <p>default value: 0.</p>	<p>Description: This specification is used in order to project external curves (curves that don't belong to the geometry to mesh) in order to constrain the mesh. External curves definition (supports) and projection tolerance are mandatory. Only relimitations of curves located below this value are projected on the geometry to mesh.</p>
--	---

CATFmtExternalCurveWeldLocalSpec

<p>This Imposed Weld Curve specification can be applied on a set of multiple curves whatever the connexity between curves may be. Specification attributes are managed using SimTopologySpecification interface:</p> <ul style="list-style-type: none"> • Geometric attribute (supports) <p>Other attributes are managed using (GetAttributeValue, SetAttributeValue):</p> <ul style="list-style-type: none"> • ReuseConnectionParameters attribute : boolean <ul style="list-style-type: none"> ◦ 1: to don't reuse connection parameters ◦ 2: to reuse connection parameters • ToleranceValue attribute : double <p>Valuate to the maximum distance between external curves and its projection on geometry to mesh.</p> <ul style="list-style-type: none"> ◦ default value: 0. <ul style="list-style-type: none"> • NbRows attribute : integer <p>Valuate the number of regular layers (row) of mesh around the edges. The possible value are {0,1,2,3}</p> <ul style="list-style-type: none"> ◦ default value: 0. <ul style="list-style-type: none"> • Height1, Height2 and Height3 attributes: double <p>Valuate respectively to the element height of the first, second and third row if specified.</p> <ul style="list-style-type: none"> ◦ Height1 default value: 0. 	<p>Description: This specification enables to project a set of curves within a geometric tolerance value. Only relimitations of curves located below this value are projected on the geometry to mesh. An uniform distribution is also created per each group of contiguous projected curves. The nodes distribution of any set of contiguous projected curves is the projection of the uniform nodes distribution (based on the mesh size value) computed on the set of the corresponding curves to be projected. Thus the number of nodes of each resulting distribution is not dependent of the projected curves. If at least one row is requested, additional regular layers of elements are created along each sides of the projected curves. To use this specification the following attributes have to be set:</p> <ul style="list-style-type: none"> • NbRow • Height1, Height2, Height3 depending on the value of NbRows: <ul style="list-style-type: none"> ◦ if NbRows is equal to 1, Height1 attribute has to be set ◦ if NbRows is equal to 2, Height1 and Height2 attributes have to be set ◦ if NbRows is equal to 3, Height1, Height2 and Height3 attributes have to be set
--	---



Local Mesh 1D Specifications

WARNING: 1D grouping specifications are the only supports possible for Local Mesh 1D Specifications. Before creating a Local Mesh 1D Specification, it is mandatory to create at least one 1D Grouping Specification with the targeted geometry supports.

CATFmtStudioDistributionSpec

Distribution specification (named for example *localSpec*) can be applied only on **one edge** resulting of an unique grouping specification.
Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- Type: integer** defining the type of distribution valued to:
 - 1: for a Uniform distribution (Isometric)
 - 2: for an Arithmetic distribution
 - 3: for a Geometric distribution
 - 4: for a distribution defined by a Law

default value: 1

- Combination: integer** indicating what are the possible combination for Arithmetic or Geometric distributions valued to:
 - 1: for a distribution defined by the number of elements (*NbElements*) and the size ratio of the elements (*Ratio21*)
 - 2: for a distribution defined by the number of elements (*NbElements*) and the size (*Size1*) at first vertex
 - 3: for a distribution defined by the number of elements (*NbElements*) and the size (*Size2*) at second vertex
 - 4: for a distribution defined by the size ratio (*Ratio21*) and the size (*Size1*) at first vertex
 - 5: for a distribution defined by the size ratio (*Ratio21*) and the size (*Size2*) at second vertex
 - 6: for a distribution defined by the size (*Size1*) at first vertex and the size (*Size2*) at second vertex

default value: 1

- CombinationUnif: integer** indicating what are the possible combination for Uniform distributions valued to:
 - 1: for a distribution defined by the number of elements (*NbElements*)
 - 2: for a distribution defined by the size (*Size1*) of the elements

default value: 1

- NbElementsValue: integer** defining the number of elements (*NbElements*)

default value: 4

- RatioValue: double** defining the ratio between the max element size and the min element size (*Ratio21*)

default value: 1.

- Size1Value: double** defining the size at first vertex (*Size1*)

default value: 0.

- Size2Value: double** defining the size at second vertex (*Size2*)

default value: 0.

- Symmetry: integer** valued to:
 - 1: to keep the distribution unchanged
 - 2: to compute a symmetric distribution with current attributes

default value: 1

Description: This specification creates a node distribution on an edge resulting of a 1D grouping specification. Depending on the choice of the distribution type (Type attribute), Combination or CombinationUnif attributes several various distributions "Option" can be computed; The next table summarize all "*Option*":

- Isometric (Uniform) distribution is defined by only one attributes(Option 1 and 2):
 - Nbelements (**CombinationUnif = 1**)
 - Size1 (**CombinationUnif = 2**)
- Non isometric distribution is defined by two attributes as indicated in the next table. There are six couples of attributes (Option 3 to 8):
 - Nbelements and Ratio21 (**Combination = 1**)
 - Nbelements and Size1 (**Combination = 2**)
 - Nbelements and Size2 (**Combination = 3**)
 - Ratio21 and Size1 (**Combination = 4**)
 - Ratio21 and Size2 (**Combination = 5**)
 - Size1 and Size2 (**Combination = 6**)

	<i>"Isometric"</i>		<i>"Arithmetic" "Geometric"</i>					
<i>"Option"</i>	1	2	3	4	5	6	7	8
<i>"NbElements"</i>	●		●	●	●			
<i>"Ratio21"</i>			●			●	●	
<i>"Size1"</i>		●		●		●		●
<i>"Size2"</i>					●	●	●	●

The next table shows examples of various distributions using additional Symmetry and Law attributes.

Examples	<i>«Arithmetic»</i>	<i>«Geometric»</i>	<i>«Uniform»</i>	<i>«Law»</i>
Attributes				
Type	<i>«Arithmetic»</i>	<i>«Geometric»</i>	<i>«Uniform»</i>	<i>«Law»</i>
NbElements	●	10	●	●
Symmetric	●	Yes	●	●
Size1	50	●	15	●
Size2	200	●	●	●
Ratio21	●	2	●	●
Reverse	●	●	●	●
Law	●	●	●	<i>y=x^2</i>

- Reverse: *integer* usable for Law and non Uniform distribution valued to:
 - 1: to keep the distribution unchanged
 - 2: to invert the distribution

default value: 1

CATFmtStudioHoleSpec

Hole specification can be applied only on **one closed edge** resulting of an unique 1D grouping specification. Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- NbElements:** *integer* valued to **the number of elements created on the contour**

default value: 3

- NbRows:** *integer* valued to the number of regular layers (row) of mesh around the contour. The possible value are {1,2,3}

default value: 1

- Height1, Height2** and **Height3:** *double* valued respectively to the element height of the first, second and third row if specified

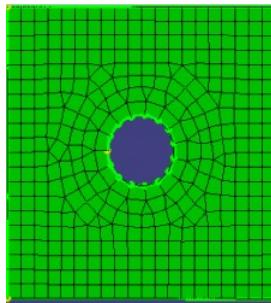
Height1 default value: 0.

Height2 default value: 0.

Height3 default value: 0.

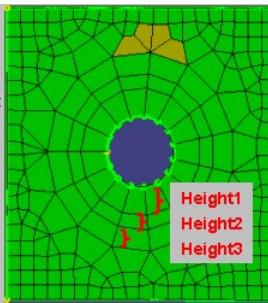
Description: This specification creates an uniform node distribution on a internal closed contour resulting of a 1D grouping specification and to get a regular mesh around this contour. To use this specification the following attributes have to be set:

- NbElements
- NbRows
- Height1, Height2, Height3 depending on the value of NbRows:
 - if NbRows is equal to 1, Height1 attribute has to be set
 - if NbRows is equal to 2, Height1 and Height2 attributes have to be set
 - if NbRows is equal to 3, Height1, Height2 and Height3 attributes have to be set



In the left case :

- NbElements = 15;
- NbRows: not specified (the default one is applied: 1);
- Height 1,2,3: not specified.



In the right case :

- NbElements = 15;
- NbRows = 3;
- Height 1,2,3: have been specified.

CATFmtStudioBoundaryLayerSpec

This mesh around edge specification can be applied only on any edges resulting of 1D grouping specification. Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- NbRows:** *integer* valued to the number of regular layers (row) of mesh around the edges. The possible value are {1,2,3}

default value: 1

- Height1, Height2** and **Height3:** *double* valued respectively to the element height of the first, second and third row if specified

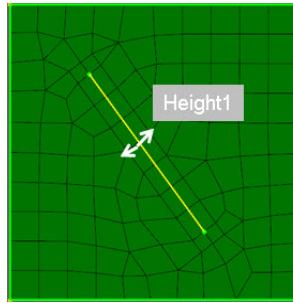
Height1 default value: 0.

Height2 default value: 0.

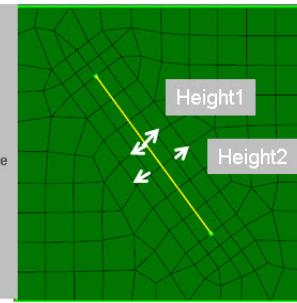
Height3 default value: 0.

Description: This specification creates an uniform node distribution on each side of the edges (free or not) resulting of a 1D grouping specification and to get a regular mesh around these. To use this specification the following attributes have to be set:

- NbRows
- Height1, Height2, Height3 depending on the value of NbRows:
 - if NbRows is equal to 1, Height1 attribute has to be set
 - if NbRows is equal to 2, Height1 and Height2 attributes have to be set
 - if NbRows is equal to 3, Height1, Height2 and Height3 attributes have to be set



In the left picture :
One layer of elements has been specified



In the right picture :
Two layers of elements have been specified

CATFmtStudioCaptureSpec

Capture specification can be applied only on **one edge** resulting of an unique 1D grouping

Description: This specification captures external mesh in order to create compatible meshes with or without geometrically coinciding or duplicated nodes.

specification. Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

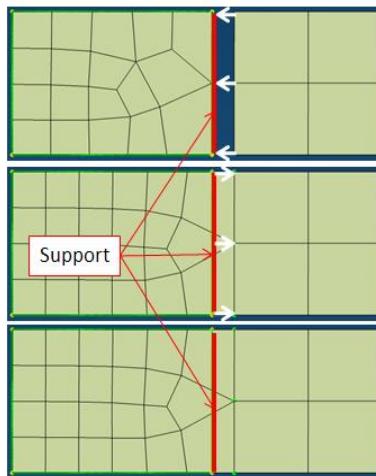
Other attributes are managed using methods (GetAttributeValue, SetAttributeValue):

- **ManualCaptureMeshMode: integer** valued to the capture mode (see picture for detailed explanations) with possible values {1,2,3}

default value: 1

- **ManualCaptureMeshTolValue: double** valued to the maximum distance under which external mesh is captured.

default value: 0.



Case 1: ManualMeshCaptureMode attribute = 1
External captured nodes are projected to define a distribution on the new mesh part.

Case 2: ManualMeshCaptureMode attribute = 2
The behavior is the same than in case 1. But new created nodes are moved at the location of captured nodes (coinciding nodes).

Case 3: ManualMeshCaptureMode attribute = 3
The behavior is the same than in case 2.
But nodes are merged with their coincident captured nodes (no more coinciding nodes).

CATFmtStudioPropagationSpec

Propagation specification can be applied on a set edges resulting of one or more 1D grouping specifications (supports). Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

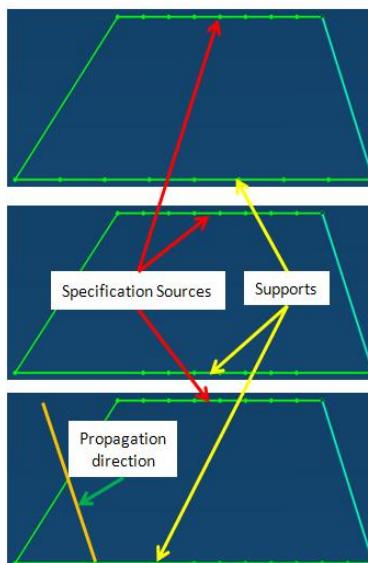
Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- **PropagationMode: integer** valued to the propagation mode (see picture for detailed explanations) with possible values {1,2,3}

default value: 1

Description: This specification propagates one or more 1D mesh local specifications (specification sources) on several edges (target supports) resulting of one or more 1D grouping specifications. PropagationMode attribute indicates how propagation is computed:

- if **PropagationMode** is equal to 1 (proportional mode), propagation is computed proportionally (see picture)
- if **PropagationMode** is equal to 2 (projection mode), propagation is computed by projection (see picture)
- if **PropagationMode** is equal to 3 (directional mode), propagation is computed following a direction given by a geometry. This geometry is managed using following method of **SimMeshPart** object (with iAttribute="Geometry"):
 - SetExternalReferences



Case 1: PropagationMode attribute = 1
Propagation is computed proportionally from the sources to the target supports.

Case 2: PropagationMode attribute = 2
Propagation is computed by projecting sources nodes on the target supports

Case 3: PropagationMode attribute = 3
Propagation is computed to keep a given direction between sources nodes and propagated nodes

Local Mesh 2D Specifications

WARNING: 2D grouping specifications are the only supports possible for Local Mesh 2D Specifications. It is therefore mandatory to create before one or more grouping 2D specifications with the targeted geometry supports.

CATFmtStudioFrontQuadSpec

Frontal Quad 2D mesh specification can be applied on a set of domains resulting of one or more 2D grouping specifications. Specification attributes are managed using **SimMeshSpecification** object:

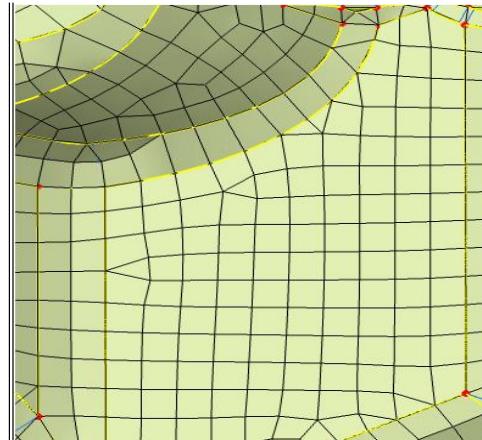
- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

Description: This specification provides quadrangle dominant elements meshes on one or more domains defined by a list of 2D grouping specifications.

- **SizeValue:** *double* valued to the mesh size

default value: 10.



CATFmtStudioFrontTriaSpec

Frontal Tria 2D mesh specification can be applied on a **set of domains** resulting of one or more 2D grouping specifications. Specification attributes are managed using **SimMeshSpecification** object:

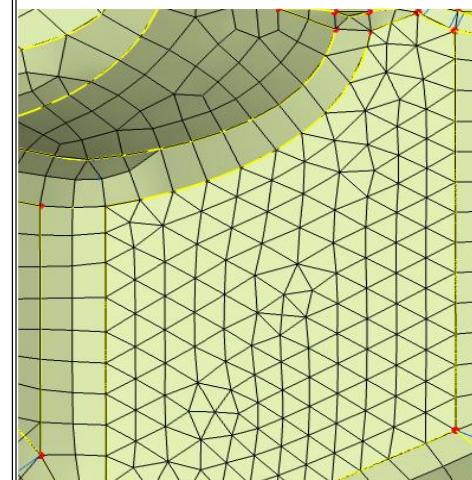
- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- **SizeValue:** *double* valued to the mesh size

default value: 10.

Description: This specification provides full triangle elements meshes on one or more domains defined by a list of 2D grouping specifications.



CATFmtStudioMappedSpec

Mapped 2D mesh specification can be applied on **one domain** resulting of **only one 2D grouping specification**. Specification attributes are managed using **SimMeshSpecification** object:

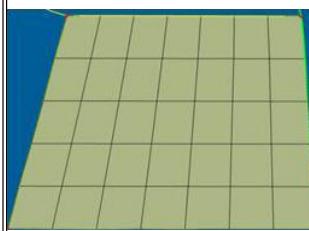
- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- **SizeValue:** *double* valued to the mesh size

default value: 10.

Description: This specification provides mapped mesh with quadrangle or triangle elements on one domain with quadrangular shape (four corners) defined by only one of 2D grouping specification.



CATFmtStudioMappedFreeMinimalSpec

Minimal mesh specification can be applied on **only one domain** resulting of **one 2D grouping specification**. Specification attributes are managed using **SimMeshSpecification** object:

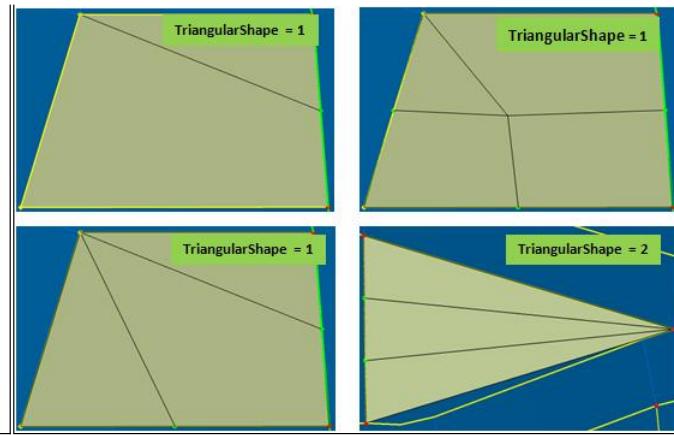
- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- **TriangularShape:** *integer* valued to the type of shape of the domain to be meshed:
 - 1: the domain to mesh has a quadrangular shape
 - 2: the domain to mesh has a triangular shape

Description: This specification provides minimal mesh on one domain with quadrangular or triangular shape (three or four corners) defined by one of 2D grouping specification. Minimal mesh is computed with the minimum of nodes on the domain boundary (a node on each existing nodes or vertices of the topology and a mesh edge between two nodes). **The mesh is not dependent of any mesh size.**

default value: 1



CATFmtStudioMappedFreeSpec

Mapped Free mesh specification can be applied on **only one domain** resulting of **only one 2D grouping specification**. Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

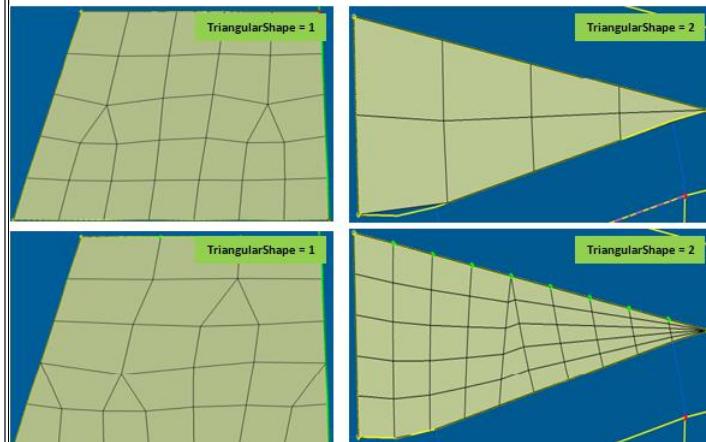
- **SizeValue: double** valuated to the mesh size

default value: 10.

- **TriangularShape: integer** valuated to the type of shape of the domain to be meshed:
 - 1: the domain to mesh has a quadrangular shape
 - 2: the domain to mesh has a triangular shape

default value: 1

Description: This specification provides mapped free mesh (regular mesh with transitional triangle elements) on one domain with quadrangular or triangular shape (three or four corners) defined by one of 2D grouping specification.



CATFmtStudioBeadSpec

Bead mesh specification can be applied on **only one domain** resulting of **only one 2D grouping specification**. Specification attributes are managed using **SimMeshSpecification** object:

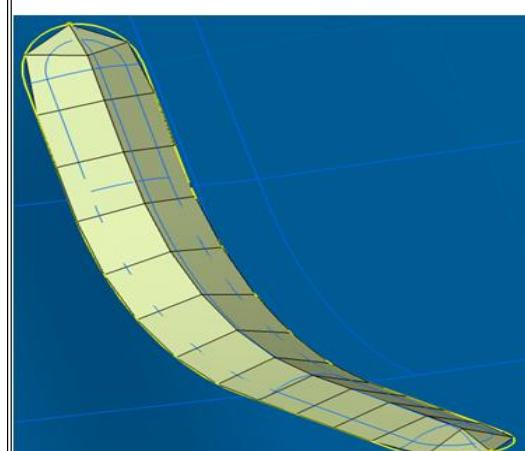
- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- **SizeValue: double** valuated to the mesh size

default value: 10.

Description: This specification provides a specific mesh for bead on one domain defined by one of 2D grouping specification. The domain must have at least two vertices on the boundary to indicate the location of triangle elements.



CATFmtStudioHalfBeadSpec

Half Bead mesh specification can be applied on **only one domain** resulting of **only one 2D grouping**

Description: This specification provides a specific mesh for half bead on one domain defined by one of 2D grouping specification. The domain must have at least three vertices on the boundary to indicate the location of triangle elements.

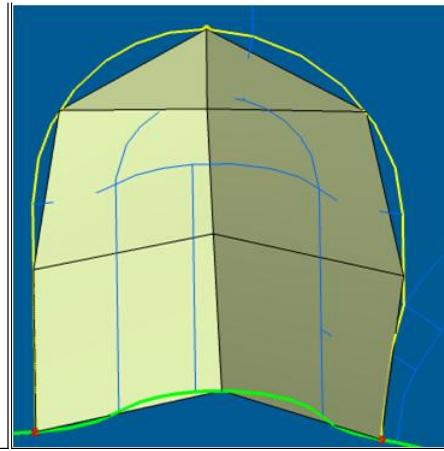
specification. Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- **SizeValue:** *double* valued to the mesh size

default value: 10.



CATFmtStudioFaceCaptureSpec

Face capture specification can be applied on a set of **external faces** containing the mesh to capture. Specification attributes are managed using **SimMeshSpecification** object.

The supports of the specification (external faces) are managed using **SimLinkAccess** object with **iName = "ConnectorList"**:

- AddLink
- RemoveLink
- GetAllLinks

Mesh to capture associated to external faces are managed using **SimLinkAccess** object with **iName= "MeshSources"**:

- AddLink
- RemoveLink
- GetAllLinks

Other attributes are managed using methods (GetAttributeValue, SetAttributeValue):

- **ManualMeshCaptureMode:** *integer* valued to the capture mode with possible values {1,2,3}

default value: 1

- **Tolerance:** *double* valued to the maximum distance under which external mesh is captured.

default value: 0.

Description: This specification captures external mesh in order to create compatible meshes with or without geometrically coinciding or duplicated nodes. The overall behavior is the following:

- if **ManualMeshCaptureMode** is equal to 1 then mesh nodes are located at the captured nodes projection on mesh.
- if **ManualMeshCaptureMode** is equal to 2 then mesh nodes are coincident with captured nodes.
- if **ManualMeshCaptureMode** is equal to 3 then mesh nodes are shared with captured nodes.

History

Version: 1 [September 2010] Document created

Surface Triangle Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Surface Triangle Mesh Part** attributes and its local specifications.

[Surface Triangle Mesh Part Creation:](#)

[Surface Triangle Mesh Part Support Definition:](#)

- [CleanedGeometry](#)
- [CleaningMode](#)

[Global Mesh Specifications:](#)

- [MeshSizeValue](#)
- [AutoMap](#)
- [ElementOrder](#)
- [AbsoluteSag](#)
- [AbsoluteSagValue](#)
- [ProportionalSag](#)
- [ProportionalSagValue](#)
- [MinCellsPerGap](#)
- [MinCellsPerGapValue](#)
- [MinMeshSize](#)
- [GrowthRatioSurface](#)
- [AutomaticMeshCapture](#)
- [AutomaticMeshCaptureTolValue](#)

Global Topology Specifications:

- [AngleBetweenCurves](#)
- [AngleBetweenFaces](#)
- [CurvatureAngle](#)
- [OffsetValue](#)
- [OffsetFromThickness](#)
- [AutomaticCurveCapture](#)
- [GeometrySimplification](#)
- [GeometrySimplificationValue](#)
- [2DHolesSuppression](#)
- [2DHolesMaxDiameter](#)
- [LogosSuppression](#)
- [LogosMaxHeight](#)
- [LogosMaxSize](#)

Local Specifications:**Local Topology Specifications:**

- [CATFmtConstrainEdgeLocalSpec](#) (Constrain Edge)
- [CATFmtUnconstrainEdgeLocalSpec](#) (Unconstrain Edge)
- [CATFmtConstrainHoleLocalSpec](#) (Constrain Hole)
- [CATFmtUnconstrainHoleLocalSpec](#) (Unconstrain Hole)
- [CATFmtGroupingLocalSpec](#) (Group Geometries)
- [CATFmtExternalPointLocalSpec](#) (Imposed Point):
 - [ToleranceValue](#)
 - [ProjectOnGeometry](#)
- [CATFmtExternalCurveLocalSpec](#) (Imposed Curve):
 - [ToleranceValue](#)

Local Mesh 1D Specifications:

- [CATFmtStudioDistributionSpec](#) (Edge Distribution):
 - [Type](#)
 - [Combination](#)
 - [CombinationUnif](#)
 - [NbElementsValue](#)
 - [RatioValue](#)
 - [Size1Value](#)
 - [Size2Value](#)
 - [Symmetry](#)
 - [Reverse](#)
- [CATFmtStudioHoleSpec](#) (Edge Distribution around Hole):
 - [NbElements](#)
 - [NbRows](#)
 - [Heighth1](#)
 - [Heighth2](#)
 - [Heighth3](#)
- [CATFmtStudioCaptureSpec](#) (Element Capture):
 - [ManualMeshCaptureMode](#)
 - [ManualMeshCaptureTolValue](#)
- [CATFmtStudioPropagationSpec](#) (Distribution Propagation):
 - [PropagationMode](#)
 - [Geometry](#)

Local Mesh 2D Specifications:

- [CATFmtLocalSpecLocalSize](#) (Local Mesh Size)
- [CATFmtStudioFrontTriaSpec](#) (Frontal Triangle Method):
 - [SizeValue](#)
- [CATFmtStudioMappedSpec](#) (Mapped Method):
 - [SizeValue](#)
- [CATFmtStudioFaceCaptureSpec](#) (Face Capture):
 - [MeshSources](#)
 - [Tolerance](#)
 - [ManualMeshCaptureMode](#)

Surface Triangle Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Surface Triangle Mesh Part, the late type is "[CATFmtSurfTriaMesher](#)".

The Surface Triangle Mesh Part can be created by using **Add** method of **SimMeshParts** object.

Surface Mesh Part Support Definition

The Surface Triangle Mesh Part support can be initialized and managed using the **AddSupport** method of **SimMeshPart** object:

- **AddSupport (iSupport As AnyObject)** to add a new support to the mesh part

CleanedGeometry

Surface Triangle Mesh Part support can be optionally modified to keep a subset of the previous geometry. For that purpose, the list of geometry to keep or to remove has to be managed using the following methods with **iSetType = "Topology"** and **iAttribute = "CleanedGeometry"**:

- **SetExternalReference (iSetType As String, iAttribute As String, iSupport As AnyObject)** to add external reference to the mesh part
- **RemoveExternalReference (iSetType As String, iAttribute As String, iSupport As AnyObject)** to remove external reference from the mesh part

Other mesh part attributes support definition are managed using following methods:

- **SetAttributeValue** (*iSetType As String*, *iAttribute As String*, *iValue*) to set a mesh attribute with an integer value
- **GetAttributeValue** (*iSetType As String*, *iAttribute As String*, *oValue*) to get a mesh attribute with an integer value

where **iType** is the type of specification and **iType** = "Topology" for support definition and **iName** the attribute name.

CleaningMode

type: <i>integer</i> valued to:	
<ul style="list-style-type: none"> • 1: to exclude geometry of CleanedGeometry of support • 2: to keep only geometry of CleanedGeometry as support 	Attribute which specifies if the geometries stored in CleanedGeometry external references have to be removed from the mesh part support definition or if this geometries will replace the initial support.
default value: 1	

Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods:

- **SetAttributeValue** (*iSetType As String*, *iAttribute As String*, *iValue*) to set a mesh attribute with an integer value
- **GetAttributeValue** (*iSetType As String*, *iAttribute As String*, *oValue*) to get a mesh attribute with an integer value

iType is the type of specification and **iType** = "Mesh" for Global Mesh Specifications.

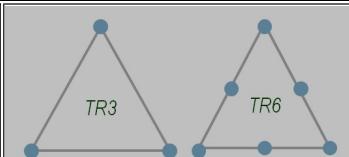
MeshSizeValue

type: <i>double</i> valued to mesh elements size.	
default value: 10.	Attribute which specifies the mesh element target size that surface mesher tries to respect.

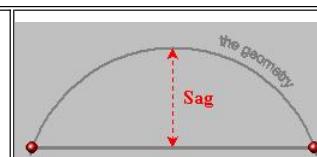
AutoMap

type: <i>integer</i> valued to:	
<ul style="list-style-type: none"> • 1: do not map mesh the support • 2: map mesh the support where appropriate 	This attribute specifies whether map mesh is applied on support where appropriate.
default value: 2	

ElementOrder

type: <i>integer</i> valued to:		
<ul style="list-style-type: none"> • 1: to get a mesh with linear elements • 2: to get a mesh with quadratic elements 	Attribute which specifies the number of intermediate nodes per element edge , either linear TR3 (on the left picture) or quadratic TR6 (on the right picture).	
default value: 1		

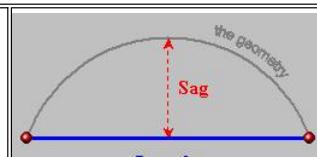
AbsoluteSag

type: <i>integer</i> valued to:		
<ul style="list-style-type: none"> • 1: to create a mesh that does not respect sag • 2: to create a mesh that respects sag 	Attribute which specifies if the mesh is created with a maximum imposed sag.	
default value: 1		

AbsoluteSagValue

type: <i>double</i> valued to the target maximum sag when AbsoluteSag attribute is valued to 2.	
default value: 1.0	Attribute which specifies the maximum value of the absolute sag (distance between the geometry of the mesh). WARNING: It is taken into account only if the AbsoluteSag attribute is equal to 2

ProportionalSag

type: <i>integer</i> valued to:		
<ul style="list-style-type: none"> • 1: to create a mesh that does not respect proportional sag • 2: to create a mesh that respects proportional sag 	Attribute which specifies if the mesh is created with a imposed maximum proportional sag which is the ratio of the Sag by the Length of a mesh edge element. (Sag/Length).	
default value: 1		

ProportionalSagValue

type: <i>double</i> valued to target proportional sag when ProportionalSag attribute is valued to 2. default value: 0.2	Attribute which specifies the maximum value of the proportional sag. WARNING: It is taken into account only if the ProportionalSag attribute is equal to 2
--	---

MinMeshSize

type: <i>double</i> valued to target minimum size for elements when AbsoluteSag or ProportionalSag attributes are valued to 2. default value: 0.	Attribute which specifies the minimum size when mesh is computed with absolute sag or proportional sag. In that cases, the mesh minimum size can be very small regarding the target MeshSizeValue and it becomes necessary to control minimum size in order to prevent to have too many elements WARNING: this attribute is taken into account only if AbsoluteSag or ProportionalSag attribute is equal to 2.
---	--

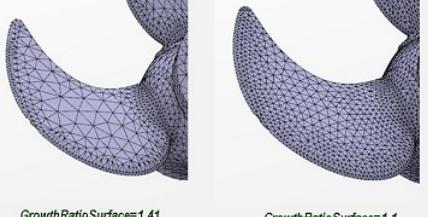
AutomaticMeshCapture

type: <i>integer</i> valued to: • 1: to ignore automatic mesh capture • 2: to compute automatic mesh capture default value: 1	Attribute which specifies if external meshes must be captured. This attribute is linked to the attribute AutomaticMeshCaptureToValue .
--	---

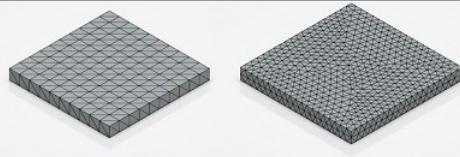
AutomaticMeshCaptureToValue

type: <i>double</i> valued to the maximum distance under which mesh is captured. default value: 0.	Attribute which specifies the maximum distance for mesh capture: if a mesh (element edges) is farthest from the geometry than this tolerance, it will not be captured even if the AutomaticMeshCapture attribute is valued to 2. WARNING: to precise exactly the list of mesh parts to capture, the method SetMeshPartsToCapture of SimMeshPart object can be used: • SetMeshPartsToCapture () To retrieve the parents mesh parts another method of SimMeshPart object can be used: • Parent ()
---	--

GrowthRatioSurface

type: <i>double</i> valued to the mesh size growth. default value: 1.41	Attribute which controls the mesh size growth between neighboring elements on the surface WARNING: It is taken into account only if the AbsoluteSag or ProportionalSag or MinCellsPerGap attribute is equal to 2	
--	---	--

MinCellsPerGap

type: <i>integer</i> valued to: • 1:to keep the default mesh • 2:to refine the mesh default value: 1	Attribute which specifies the mesh refinement to be applied in narrow regions.	
---	--	--

MinCellsPerGapValue

type: <i>integer</i> valued to the expected number of layers across the thickness. default value: 1	Attribute which specifies the number of expected layers across the thickness. The mesh is refined (see picture above). WARNING: It is taken into account only if the MinCellsPerGap attribute is equal to 2.
--	---

Global Topology Specifications

Global topology specifications can be set or retrieved using following methods:

- **SetAttributeValue** (*iSetType As String*, *iAttribute As String*, *iValue*) to set a topology attribute
- **GetAttributeValue** (*iSetType As String*, *iAttribute As String*, *oValue*) to get a topology attribute

iType is the type of specification and **iType = "Topology"** for Global Topology Specifications.

AngleBetweenCurves

<p>type: <i>double (radian unit)</i> valuated to the angle under which vertices are inactivated default value: 20°</p>	<p>Attribute which specifies the maximum angle under which a vertex connected exactly to two constrained edges can be inactivated.</p>	
--	--	--

AngleBetweenFaces

<p>type: <i>double (radian unit)</i> valuated to the angle under which edges are inactivated. default value: 20°</p>	<p>Attribute which specifies the maximum angle under which an edge connected exactly to two faces can be inactivated during topology computation.</p>	
--	---	--

CurvatureAngle

<p>type: <i>double (radian unit)</i> valuated to the angle over which edges are activated. default value: 20°</p>	<p>Attribute which specifies the minimum angle computed along filleted surfaces over which edges are activated. For a given fillet, when the computed curvature angle is greater than CurvatureAngleValue all boundary edges of this fillet are activated otherwise they are not.</p>	
---	--	--

OffsetValue

<p>type: <i>double</i> valuated to the offset between the geometry and the mesh. default value: 0.</p>	<p>Attribute which specifies the distance of the offset between the mesh and the geometry.</p>	
--	--	--

OffsetFromThickness

<p>type: <i>integer</i> valuated to:</p> <ul style="list-style-type: none"> • 1: if OffsetValue has not been computed from geometry thickness • 2: if OffsetValue has been computed from geometry thickness <p>default value: 0.</p>	<p>Attribute which specifies if OffsetValue has been computed from geometry thickness using for example CATIGSMThickness interface.</p>
--	--

AutomaticCurveCapture

<p>type: <i>integer</i> valuated to:</p> <ul style="list-style-type: none"> • 1: to ignore automatic capture of external curves • 2: to activate automatic capture of external curves <p>default value: 1</p>	<p>Attribute which specifies if external curves must be captured automatically. It is possible to automatically capture curve only when AutomaticMeshCapture attribute value is set to 2 (enabled). The geometries used to compute the projections are the support of the mesh part to be captured. The tolerance used to compute the projections is AutomaticMeshCaptureTolValue.</p>
---	--

GeometrySimplification

--

type: *integer* valued to:

- 1: to keep grouping topology without additional simplification
- 2: to compute topology simplification

default value: 2

Attribute which specifies if topology simplification is computed. Topology simplification objective is to minimize the number of bad elements regarding the Min Height element quality criterion.

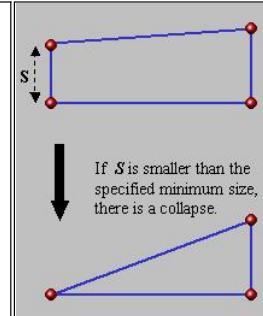
GeometrySimplificationValue

type: *double* valued to the Min Height criterion targeted for mesh elements.

default value: $0.1 * \text{MeshSizeValue}$

When topology simplification is computed, different topology modifications are done automatically using a set of topological operators in order to respect Min Height element quality criterion:

- Vertex Creation/Activation/Inactivation
- Edge Creation/Activation/Inactivation
- Edge collapse (see picture)
- Edge Merge



2DHolesSuppression

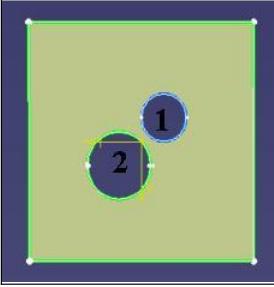
type: *integer* valued to:

- 1: to ignore hole control suppression
- 2: to control hole suppression

default value: 1

Attribute which specifies whether hole suppression will be done. Hole suppression is computed from hole diameter value.

For example in the right picture, a suppress hole size (**2DHolesMaxDiameter** attribute) was specified for the mesh part. Hole 1 diameter is below than this value and was ignored conversely to Hole 2.



2DHolesMaxDiameter

type: *double* valued to the maximum diameter under which a hole is ignored.

default value: 0.

Attribute which specifies the maximum diameter allowed for hole suppression. This value is computed as the maximum distance between two points of a hole discretization.

WARNING: It is taken into account only if the **2DHolesSuppression** attribute is equal to 2.

LogosSuppression

type: *integer* valued to:

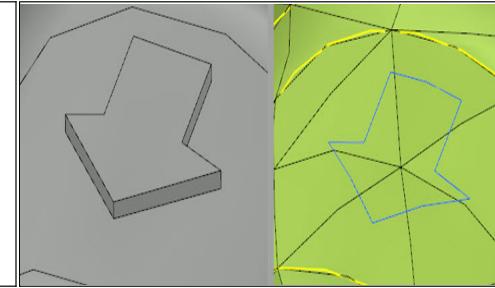
- 1: to ignore logos control suppression
- 2: to control logos suppression

default value: 1

Attribute which specifies whether logos suppression will be done.

Logos suppression consists of ignoring small features of the geometry that are useless for the mesh (small height regarding the mesh size). Logos suppression is driven by two attributes: **LogosMaxHeight** and **LogosMaxSize**.

For example on the right picture, an ignored logo can be seen.



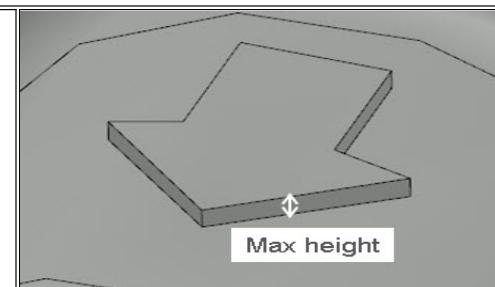
LogosMaxHeight

type: *double* valued to the max height of a detectable logo

default value: $0.1 * \text{MeshSizeValue}$

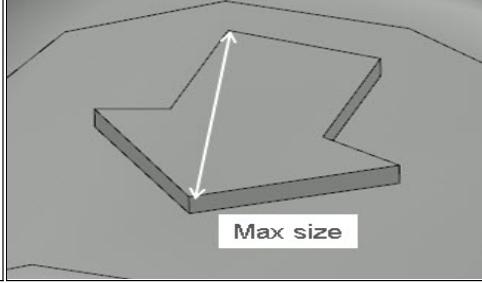
Attribute which specifies the maximal height allowed to detect logos.

The maximum height of a logo can be defined as the maximum distance of any points on the logo faces to the extrapolated bounding surfaces.



LogosMaxSize

--	--	--

<p>type: <i>double</i> valued to the max size of a detectable logo</p> <p>default value: <i>MeshSizeValue</i></p>	<p>Attribute which specifies the maximal size allowed to detect logos.</p> <p>The maximum size of a logo can be defined as the maximum distance between two points on the logo boundary (maximum diameter).</p>	
---	---	--

Local Specifications

The Surface Triangle Mesh Part has two kind of local specifications: **topology** and **mesh** local specifications. These specifications are managed by collection object:

- **SimMeshSpecifications:** for the collection of local mesh specifications
- **SimTopologySpecifications:** for the collection of local topology specifications

You should use the following methods of **SimMeshPart** object to retrieve these collections:

- **GetMeshSpecifications** to retrieve the local mesh specification collection
- **GetTopologySpecification** to retrieve the local topology specification collection

Local specifications are managed from the collection objects:

- **Add (iType As String)** to create a new local specification with:
 - iType is the late type of the specification to create
- **Item (iIndex)** to retrieve a local specification with:
 - iIndex is the index of the local specification in its collection
- **Remove (iIndex)** to delete a local specification with:
 - iIndex is the index of the local specification in its collection

Attribute of local specification are set using following method from **SimMeshSpecification** or **SimTopologySpecification** object depending of the kind of local specification:

- **SetAttributeValue (iAttribute As String, iValue)** to set a local specification attribute value

In a manner similar, attribute of local specification are retrieved using following method:

- **GetAttributeValue (iAttribute As String, iValue)** to retrieve a local specification

Geometric attribute of local specification (supports) are managed using **SimLinkAccess** object:

- **AddLink (iName As String, iLink As AnyObject)**
- **RemoveAllLinks (iName As String)**
- **RemoveLink (iName As String, iLink As AnyObject)**
 - iName is the name of the link container, here it's "**ConnectorList**"

Local Topology Specifications

CATFmtConstrainEdgeLocalSpec

<p>Specification attributes are managed using SimTopologySpecification object.</p> <ul style="list-style-type: none"> • Only geometric attribute (supports) 	<p>Description: This specification is used in order to constrain a set of edges resulting of one or more 1D grouping specifications.</p>
---	--

CATFmtUnconstrainEdgeLocalSpec

<p>Specification attributes are managed using SimTopologySpecification object.</p> <ul style="list-style-type: none"> • Only geometric attribute (supports) 	<p>Description: This specification is used in order to unconstrain a set of edges resulting of one or more 1D grouping specifications.</p>
---	--

CATFmtConstrainHoleLocalSpec

<p>Specification attributes are managed using SimTopologySpecification object.</p> <ul style="list-style-type: none"> • Only geometric attribute (supports) 	<p>Description: This specification is used in order to constrain a hole resulting of one or more 1D grouping specifications.</p>
---	--

CATFmtUnconstrainHoleLocalSpec

<p>Specification attributes are managed using SimTopologySpecification object.</p> <ul style="list-style-type: none"> • Only geometric attribute (supports) 	<p>Description: This specification is used in order to unconstrain a hole resulting of one or more 1D grouping specifications.</p>
---	--

CATFmtGroupingLocalSpec

	<p>Description: This specification is used in order to group geometries from a set of supports (internal 1D or 2D geometries that belong to the geometry to mesh). The</p>
--	---

Specification attributes are managed using **SimTopologySpecification** object:

- Only geometric attribute (supports)

overall behavior is the following:

- if **1D geometries** have been selected the program tries to **create an edge** of the topology per group of connected selected **1D geometries**. **1D selected geometries are also constrained** during this execution.

WARNING: It is possible to select only one support which useful to set local mesh specifications

- if **2D geometries** have been selected the program tries to **create a domain** of the topology per group of connected selected **2D geometries**. Consequently, **all boundary edges of new created domains are constrained and all internal edges of new created domains are unconstrained**.

CATFmtExternalPointLocalSpec

Specification attributes are managed using **SimTopologySpecification** interface:

- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- **ToleranceValue: double** valuated to the maximum distance between external point and its projection on geometry to mesh.

default value: 0.

- **ProjectOnGeometry: integer** valuated to the target geometry choice

default value: 1

Description: This specification is used in order to project **external points** (points that don't belong to the geometry to mesh) in order to constrain the mesh. External points definition (supports) and projection tolerance are mandatory. The overall behavior is the following:

- if the **distance** between an external point and the geometry to mesh is **greater** than the **ToleranceValue**, the point will not be projected
- if the **distance** between an external point and the geometry to mesh is **lower** than the **ToleranceValue**, the point will be projected. The final location of this point depends on **ProjectOnGeometry** attribute:
 - if **ProjectOnGeometry** is equal to **1**, the **node** representing the point will be **located** exactly on the projection point on the surface geometry
 - if **ProjectOnGeometry** is equal to **2**, the **node** representing the point projection will be **located** exactly on the **external point (same Cartesian coordinates)**

CATFmtExternalCurveLocalSpec

Specification attributes are managed using **SimTopologySpecification** interface:

- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- **ToleranceValue attribute** is set and retrieved with:
 - SetAttributeValue
 - GetAttributeValue

type: double valuated to the maximum distance between external curves and its projection on geometry to mesh.

default value: 0.

Description: This specification is used in order to **project external curves** (curves that don't belong to the geometry to mesh) in order to constrain the mesh. External curves definition (supports) and projection tolerance are mandatory. Only **relimitations of curves located below this value** are projected on the geometry to mesh.

Local Mesh 1D Specifications

WARNING: 1D grouping specifications are the only supports possible for Local Mesh 1D Specifications. Before creating a **Local Mesh 1D Specification**, it is mandatory to create at least one **1D Grouping Specification** with the targeted geometry supports.

CATFmtStudioDistributionSpec

Distribution specification (named for example **localSpec**) can be applied only on **one edge** resulting of an unique grouping specification. Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- **Type: integer** defining the type of distribution valuated to:
 - **1:** for a Uniform distribution (Isometric)
 - **2:** for an Arithmetic distribution
 - **3:** for a Geometric distribution
 - **4:** for a distribution defined by a Law

default value: 1

- **Combination: integer** indicating what are the possible combination for Arithmetic or Geometric distributions valuated to:
 - **1:** for a distribution defined by the number of elements (*NbElements*) and the size ratio of the elements (*Ratio21*)
 - **2:** for a distribution defined by the number of elements (*NbElements*) and the size (*Size1*) at first vertex

Description: This specification creates a node distribution on an edge resulting of a 1D grouping specification. Depending on the choice of the distribution type (Type attribute), Combination or CombinationUnif attributes several various distributions "Option" can be computed. The next table summarize all "*Option*".

- Isometric (Uniform) distribution is defined by only one attributes(Option 1 and 2):
 - **Nbelements (CombinationUnif= 1)**
 - **Size1 (CombinationUnif= 2)**
- Non isometric distribution is defined by two attributes as indicated in the next table. There

- **3:** for a distribution defined by the number of elements (*NbElements*) and the size (*Size2*) at second vertex
- **4:** for a distribution defined by the size ratio (*Ratio21*) and the size (*Size1*) at first vertex
- **5:** for a distribution defined by the size ratio (*Ratio21*) and the size (*Size2*) at second vertex
- **6:** for a distribution defined by the size (*Size1*) at first vertex and the size (*Size2*) at second vertex

default value: 1

- **CombinationUnif:** *integer* indicating what are the possible combination for Uniform distributions valued to:
 - **1:** for a distribution defined by the number of elements (*NbElements*)
 - **2:** for a distribution defined by the size (*Size1*) of the elements

default value: 1

- **NbElementsValue:** *integer* defining the number of elements (*NbElements*)

default value: 4

- **RatioValue:** *double* defining the ratio between the max element size and the min element size (*Ratio21*)

default value: 1.

- **Size1Value:** *double* defining the size at first vertex (*Size1*)

default value: 0.

- **Size2Value:** *double* defining the size at second vertex (*Size2*)

default value: 0.

- **Symmetry:** *integer* valued to:
 - **1:** to keep the distribution unchanged
 - **2:** to compute a symmetric distribution with current attributes

default value: 1

- **Reverse:** *integer* usable for Law and non Uniform distribution valued to:
 - **1:** to keep the distribution unchanged
 - **2:** to invert the distribution

default value: 1

are six couples of attributes (Option 3 to 8):

- *Nbelements* and *Ratio21* (**Combination = 1**)
- *NbElements* and *Size1* (**Combination = 2**)
- *NbElements* and *Size2* (**Combination = 3**)
- *Ratio21* and *Size1* (**Combination = 4**)
- *Ratio21* and *Size2* (**Combination = 5**)
- *Size1* and *Size2* (**Combination = 6**)

	<i>"Isometric"</i>		<i>"Arithmetic" \ "Geometric"</i>					
<i>"Option"</i>	1	2	3	4	5	6	7	8
<i>"NbElements"</i>	●		●	●	●			
<i>"Ratio21"</i>			●			●	●	
<i>"Size1"</i>		●		●		●		●
<i>"Size2"</i>				●		●	●	●

The next table shows examples of various distributions using additional Symmetry and Law attributes.

Examples	1	2	3	4
Attributes				
Type	«Arithmetic»	«Geometric»	«Uniform»	«Law»
NbElements	●		●	●
Symmetric	●	Yes	●	●
Size1	50	●	15	●
Size2	200	●	●	●
Ratio21	●	2	●	●
Reverse	●	●	●	●
Law	●	●	●	y=x ²

CATFmtStudioHoleSpec

Hole specification can be applied only on **one closed edge** resulting of an unique 1D grouping specification. Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- **NbElements:** *integer* valued to **the number of elements created on the contour**

default value: 3

- **NbRows:** *integer* valued to the number of regular layers (row) of mesh around the contour. The possible value are {1,2,3}

default value: 1

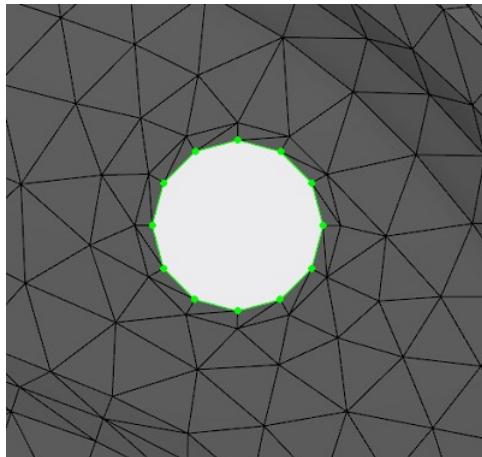
- **Height1, Height2 and Height3:** *double* valued respectively to the element height of the first, second and third row if specified

Height1 default value: 0.

Height2 default value: 0.

Height3 default value: 0.

Description: This specification creates an uniform node distribution on a internal closed contour resulting of a 1D grouping specification and to get a regular mesh around this contour.



CATFmtStudioCaptureSpec

Capture specification can be applied only on **one edge** resulting of an unique 1D grouping

specification. Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

Other attributes are managed using methods (GetAttributeValue, SetAttributeValue):

- **ManualCaptureMeshMode: integer** valued to the capture mode with possible values {1,2,3}

default value: 1

- **ManualCaptureMeshTolValue: double** valued to the maximum distance under which external mesh is captured.

default value: 0.

Description: This specification captures external mesh in order to create compatible meshes with or without geometrically coinciding or duplicated nodes. The overall behavior is the following:

- if **ManualCaptureMeshMode** is equal to 1 then mesh nodes are located at the captured nodes projection on mesh.
- if **ManualCaptureMeshMode** is equal to 2 then mesh nodes are coincident with captured nodes.
- if **ManualCaptureMeshMode** is equal to 3 then mesh nodes are shared with captured nodes.

CATFmtStudioPropagationSpec

Propagation specification can be applied on a set edges resulting of one or more 1D grouping specifications (supports). Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

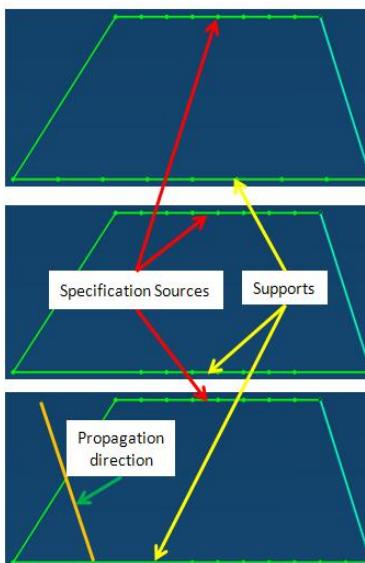
Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- **PropagationMode: integer** valued to the propagation mode (see picture for detailed explanations) with possible values {1,2,3}

default value: 1

Description: This specification propagates one or more 1D mesh local specifications (specification sources) on several edges (target supports) resulting of one or more 1D grouping specifications. PropagationMode attribute indicates how propagation is computed:

- if **PropagationMode** is equal to 1 (proportional mode), propagation is computed proportionally (see picture)
- if **PropagationMode** is equal to 2 (projection mode), propagation is computed by projection (see picture)
- if **PropagationMode** is equal to 3 (directional mode), propagation is computed following a direction given by a geometry. This geometry is managed using following method of **SimMeshPart** object (with **IAttribute="Geometry"**):
 - SetExternalReferences



Local Mesh 2D Specifications

WARNING: Except for CATFmtLocalSpecLocalSize which works with geometrical elements as an input, **2D grouping specifications are the only supports possible for all the other Local Mesh 2D Specifications**. It is therefore **mandatory to create before one or more grouping 2D specifications with the targeted geometry supports**.

CATFmtLocalSpecLocalSize

Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

Other attributes are managed using (GetAttributeValue, SetAttributeValue):

- **SizeValue: double** valued to the target size on the support
- **AbsoluteSag: int** valued to
 - 1: to specify only a target mesh size (default)
 - 2: to specify a local sag as well
- **AbsoluteSagValue: double** valued to the target sag on the support (only taken into

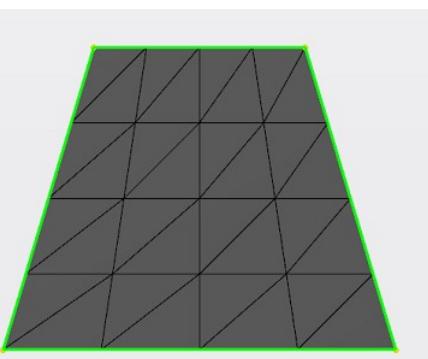
Description: This specification is used in order to specify a local mesh size or sag on a set of faces.

account if the **AbsoluteSag** attribute above is set to 2)

CATFmtStudioFrontTriaSpec

<p>Frontal Tria 2D mesh specification can be applied on a set of domains resulting of one or more 2D grouping specifications. Specification attributes are managed using SimMeshSpecification object:</p> <ul style="list-style-type: none"> • Geometric attribute (supports) <p>Other attributes are managed using (GetAttributeValue, SetAttributeValue):</p> <ul style="list-style-type: none"> • SizeValue: <i>double</i> valued to the mesh size <p>default value: 10.</p>	<p>Description: This specification provides full triangle elements meshes on one or more domains defined by a list of 2D grouping specifications.</p>
---	--

CATFmtStudioMappedSpec

<p>Mapped 2D mesh specification can be applied on one domain resulting of only one 2D grouping specification. Specification attributes are managed using SimMeshSpecification object:</p> <ul style="list-style-type: none"> • Geometric attribute (supports) <p>Other attributes are managed using (GetAttributeValue, SetAttributeValue):</p> <ul style="list-style-type: none"> • SizeValue: <i>double</i> valued to the mesh size <p>default value: 10.</p>	<p>Description: This specification provides mapped mesh with triangle elements on one domain with quadrangular shape (four corners) defined by only one of 2D grouping specification.</p> 
--	--

CATFmtStudioFaceCaptureSpec

<p>Face capture specification can be applied on a set of external faces containing the mesh to capture. Specification attributes are managed using SimMeshSpecification object.</p> <p>The supports of the specification (external faces) are managed using SimLinkAccess object with iName = "ConnectorList":</p> <ul style="list-style-type: none"> • AddLink • RemoveLink • GetAllLinks <p>Mesh to capture associated to external faces are managed using SimLinkAccess object with iName = "MeshSources":</p> <ul style="list-style-type: none"> • AddLink • RemoveLink • GetAllLinks <p>Other attributes are managed using methods (GetAttributeValue, SetAttributeValue):</p> <ul style="list-style-type: none"> • ManualMeshCaptureMode: <i>integer</i> valued to the capture mode with possible values {1,2,3} <p>default value: 1</p> <ul style="list-style-type: none"> • Tolerance: <i>double</i> valued to the maximum distance under which external mesh is captured. <p>default value: 0.</p>	<p>Description: This specification captures external mesh in order to create compatible meshes with or without geometrically coinciding or duplicated nodes. The overall behavior is the following:</p> <ul style="list-style-type: none"> • if ManualMeshCaptureMode is equal to 1 then mesh nodes are located at the captured nodes projection on mesh. • if ManualMeshCaptureMode is equal to 2 then mesh nodes are coincident with captured nodes. • if ManualMeshCaptureMode is equal to 3 then mesh nodes are shared with captured nodes.
---	--

History

Version: 1 [Oct 2014] Document created

Sweep 3D Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Sweep 3D Mesh Part** attributes and its local specifications.

Sweep 3D Mesh Part Creation**Sweep 3D Mesh Part Support Definition:**

- [Source](#)
- [Target](#)

Global Mesh Specifications:

- [ElementOrder](#)
- [NbElementsValue](#)
- [Type](#)
- [RatioValue](#)
- [Symmetry](#)
- [Automatic2DMeshCreation](#)
- [MeshSize](#)
- [AutomaticMeshCaptureTolValue](#)

Local Specifications**Local Topology Specifications:**

- [CATFmtSweep3DGeometryImposedGuide](#)
- [CATFmtSweep3DGeometryExcludedGuide](#)

Sweep 3D Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.
In the case of the sweep 3D mesh part, the late type is "CATFmtSweep3DMesher".
The sweep 3D mesh part can be created by using **Add** method of **SimMeshParts** collection object.

Sweep 3D Mesh Part Support Definition

The Sweep 3D Mesh Part support can be initialized and managed using the **AddSupport** method of **SimMeshPart** object:

- **AddSupport (iSupport As AnyObject)** to add a new support to the mesh part

Source

The Sweep 3D Mesh Part source support can be set or unset using following methods of the **SimMeshPart** object:

- **SetExternalReference (iSetType As String, iAttribute As String, iSupport As AnyObject)** to set the source face where:
 - **iSetType="Topology"**
 - **iAttribute="Source"**
 - **iSupport** is a geometric face
- **RemoveExternalReference (iSetType As String, iAttribute As String, iSupport As AnyObject)** to unset the source face where:
 - **iSetType="Topology"**
 - **iAttribute="Source"**
 - **iSupport** is a geometric face

Target

The Sweep 3D Mesh Part target support can be set or unset using following methods of the **SimMeshPart** object:

- **SetExternalReference (iSetType As String, iAttribute As String, iSupport As AnyObject)** to set the target face where:
 - **iSetType="Topology"**
 - **iAttribute="Target"**
 - **iSupport** is a geometric face
- **RemoveExternalReference (iSetType As String, iAttribute As String, iSupport As AnyObject)** to unset the target face where:
 - **iSetType="Topology"**
 - **iAttribute="Target"**
 - **iSupport** is a geometric face

WARNING: If no **Source** and **Target** support are set they will be automatically computed at mesh update.

Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods of the **SimMeshPart** object:

- **SetAttributeValue (iSetType As String, iAttribute As String, iValue)** to set a mesh attribute where:
 - **iSetType="Mesh"**
 - **iAttribute** is the attribute's name
 - **iValue** is the attribute's value
- **GetAttributeValue (iSetType As String, iAttribute As String, oValue)** to get a mesh attribute where:
 - **iSetType="Mesh"**
 - **iAttribute** is the attribute's name
 - **oValue** is the attribute's value

ElementOrder

type: integer valued to:

Attribute which specifies the number of intermediate nodes per element edge.

- 1: to get a mesh with linear hexaedron (HE8)
- 2: to get a mesh with quadratic hexaedron (HE20)

default value: 1**NbElementsValue**

type: *integer* valued to the number of sweep layers. This attribute specifies the number of sweep layers.

Type

type: *integer* valued to:

- 1: to get an uniform distribution
- 2: to get an arithmetic distribution This attribute specifies the sweep layers distribution.
- 3: to get a geometric distribution

default value: 1**RatioValue**

type: *double* valued to the size ratio between the first and last layer thicknesses.

This attribute specifies the size ratio *R* between the first layer thickness *h1* and last layer thickness *h2*.

default value: 1.0

WARNING: It is taken into account only if the **Type** attribute is equal to 2 or 3.

Symmetry

type: *integer* valued to:

- 1: to keep the distribution unchanged This attribute specifies the sweep layers distribution as symmetric or not.
- 2: to compute a symmetric distribution with current attributes **WARNING:** It is taken into account only if the **Type** attribute is equal to 2 or 3.

default value: 1**Automatic2DMeshCreation**

type: *integer* valued to:

- 1 to reuse existing mesh by capture This attribute specifies whether (intermediate) surface mesh should be generated on the Source face(s) (2) or not (1).
- 2 to generate an (intermediate) surface mesh on Source face(s)

MeshSize

Attribute which specifies the mesh element target size that mesher tries to respect on the Source face(s).

type: *double* valued to mesh elements size.

WARNING: It is taken into account only if the **Automatic2DMeshCreation** is set to 2.

AutomaticMeshCaptureToValue

Attribute which specifies the maximum distance for mesh capture: if a mesh is farthest from the geometry than this tolerance, it will not be captured.

To precise exactly the list of mesh parts to capture the method **SetMeshPartsToCapture** of **SimMeshPart** object can be used:

- **SetMeshPartsToCapture ()**

To retrieve the parents mesh parts another method of **SimMeshPart** object can be used:

- **Parent ()**

WARNING: It is taken into account only if the **Automatic2DMeshCreation** is set to 1.

Local Specifications

The local specifications are managed by collection object:

- **SimTopologySpecifications** is the collection of **SimTopologySpecification** object (local topology specification)
- **SimMeshSpecifications** is the collection of **SimMeshSpecification** object (local mesh specification)

You should use the following methods of **SimMeshPart** object to retrieve this collection:

- **GetTopologySpecification** to retrieve the local topology specification collection
- **GetMeshSpecification** to retrieve the local mesh specification collection

Local specifications are managed from the collection object:

- **Add (iType As String, oSpec)** to create a new local specification with:
 - iType is the late type of the specification to create
 - oSpec is a local specification (**SimTopologySpecification** or **SimMeshSpecification**)
- **Item (iIndex, oSpec)** to retrieve a local specification with:
 - iIndex is the index of the local specification in its collection
 - oSpec is a local specification (**SimTopologySpecification** or **SimMeshSpecification**)
- **Remove (iIndex)** to delete a local specification with:
 - iIndex is the index of the local specification in its collection

Local specification attributes are managed using the local specification object:

- **SetAttributeValue** (*iAttribute As String, iValue*) to set a local specification attribute where:
 - **iAttribute** is the attribute's name
 - **iValue** is the attribute's value
- **GetAttributeValue** (*iAttribute As String, oValue*) to get a local specification attribute where:
 - **iAttribute** is the attribute's name
 - **oValue** is the attribute's value

Geometric attribute of local specification (supports) are managed using **SimLinkAccess** object:

- **AddLink** (*iName As String, iLink As AnyObject*)
- **RemoveAllLinks** (*iName As String*)
- **RemoveLink** (*iName As String, iLink As AnyObject*)
 - **iName** is the name of the link container, here it's "**ConnectorList**"

Local Topology Specifications

CATFmtSweep3DGeometryImposedGuide

Specification attributes are managed using SimTopologySpecification object.	Description: This specification is used to impose geometric edges as sweep guides.
<ul style="list-style-type: none"> • Only geometric attribute (supports) 	

CATFmtSweep3DGeometryExcludedGuide

Specification attributes are managed using SimTopologySpecification object.	Description: This specification is used to exclude geometric edges from the sweep guides.
<ul style="list-style-type: none"> • Only geometric attribute (supports) 	

Symmetry Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Symmetry Mesh Part** attributes and its local specifications.

[Symmetry Mesh Part Creation](#)

[Symmetry Mesh Part Support Definition:](#)

- [Geometry](#)

[Global Mesh Specifications:](#)

- [AutomaticMeshCapture](#)
- [AutomaticMeshCaptureTolValue](#)

Symmetry Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Symmetry mesh part, the late type is "CATFmtSymmetryMesher".

The Symmetry mesh part can be created by using **Add** method of **SimMeshParts** collection object.

Symmetry Mesh Part Support Definition

Symmetry Mesh Part support can be set using following method of the **SimMeshPart** object:

- **AddSupport** (*iSupport As AnyObject*) to add a new support to the mesh part

Geometry

The Symmetry Mesh Part plane can be set or unset using following methods of the **SimMeshPart** object:

- **SetExternalReference** (*iSetType As String, iAttribute As String, iSupport As AnyObject*) to set the symmetry plane where:
 - **iSetType**="Mesh"
 - **iAttribute**="Geometry"
 - **iSupport** is a geometric plane
- **RemoveExternalReference** (*iSetType As String, iAttribute As String, iSupport As AnyObject*) to unset the symmetry plane where:
 - **iSetType**="Mesh"
 - **iAttribute**="Geometry"
 - **iSupport** is a geometric plane

Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods of the **SimMeshPart** object:

- **SetAttributeValue** (*iSetType As String, iAttribute As String, iValue*) to set a mesh attribute where:
 - **iSetType**="Mesh"
 - **iAttribute** is the attribute's name
 - **iValue** is the attribute's value
- **GetAttributeValue** (*iSetType As String, iAttribute As String, oValue*) to get a mesh attribute where:
 - **iSetType**="Mesh"
 - **iAttribute** is the attribute's name

- o **oValue** is the attribute's value

AutomaticMeshCapture

type: <i>integer</i> valuated to:	
<ul style="list-style-type: none"> • 1: to ignore automatic mesh capture • 2: to compute automatic mesh capture 	Attribute which specifies if external meshes must be captured. This attribute is linked to the attribute AutomaticMeshCaptureToValue .
default value: 1	

AutomaticMeshCaptureToValue

type: <i>double</i> valuated to the maximum distance under which mesh is captured.	Attribute which specifies the maximum distance for mesh capture: if a mesh (element edges) is farthest from the support than this tolerance, it will not be captured even if the AutomaticMeshCapture attribute is valuated to 2.
default value: 0.	<p>WARNING: to precise exactly the list of mesh parts to capture, the method SetMeshPartsToCapture of SimMeshPart object can be used:</p> <ul style="list-style-type: none"> • SetMeshPartsToCapture () <p>To retrieve the parents mesh parts another method of SimMeshPart object can be used:</p> <ul style="list-style-type: none"> • Parent ()

History

Version: 1 [October 2014] Document created

Symmetry Extrusion Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Symmetry Extrusion Mesh Part** attributes and its local specifications.

[Symmetry Extrusion Mesh Part Creation](#)

[Symmetry Extrusion Mesh Part Support Definition:](#)

- [Geometry](#)

[Global Mesh Specifications:](#)

- [Type](#)
- [RatioValue](#)
- [NbElementsValue](#)
- [Symmetry](#)
- [AutomaticMeshCapture](#)
- [AutomaticMeshCaptureToValue](#)

Symmetry Extrusion Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Symmetry Extrusion mesh part, the late type is "CATFmtExtrusionSymmetryMesher".

The Symmetry Extrusion mesh part can be created by using **Add** method of **SimMeshParts** object.

Symmetry Extrusion Mesh Part Support Definition

Symmetry Extrusion Mesh Part support can be set using following method of the **SimMeshPart** object:

- **AddSupport (iSupport As AnyObject)** to add a new support to the mesh part

Geometry

The Symmetry Extrusion Mesh Part plane can be set or unset using following methods of the **SimMeshPart** object:

- **SetExternalReference (iSetType As String, iAttribute As String, iSupport As AnyObject)** to set the symmetry plane where:
 - o **iSetType="Mesh"**
 - o **iAttribute="Geometry"**
 - o **iSupport** is a geometric plane
- **RemoveExternalReference (iSetType As String, iAttribute As String, iSupport As AnyObject)** to unset the symmetry plane where:
 - o **iSetType="Mesh"**
 - o **iAttribute="Geometry"**
 - o **iSupport** is a geometric plane

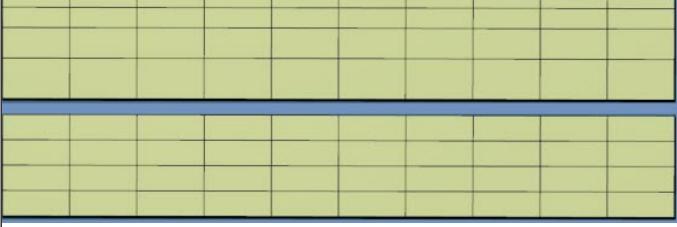
Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods of the **SimMeshPart** object:

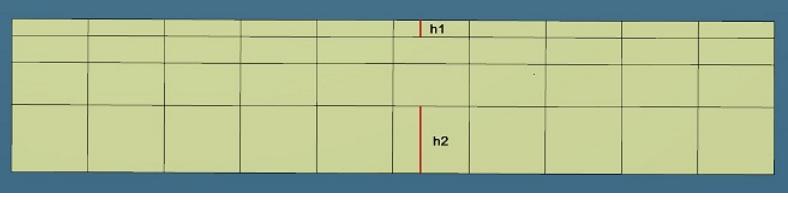
- **SetAttributeValue (iSetType As String, iAttribute As String, iValue)** to set a mesh attribute where:
 - o **iSetType="Mesh"**

- **iAttribute** is the attribute's name
- **iValue** is the attribute's value
- **GetAttributeValue** (*iSetType As String, iAttribute As String, oValue*) to get a mesh attribute where:
 - **iSetType="Mesh"**
 - **iAttribute** is the attribute's name
 - **oValue** is the attribute's value

Type

type: integer valued to:	This attribute specifies the size distribution to be applied to the mesh.	
• 1: to get an uniform distribution • 2: to get an arithmetic distribution • 3: to get a geometric distribution	The picture shows how a uniform distribution (at the bottom) differs from a geometric one (at the top, using a ratio of 3).	

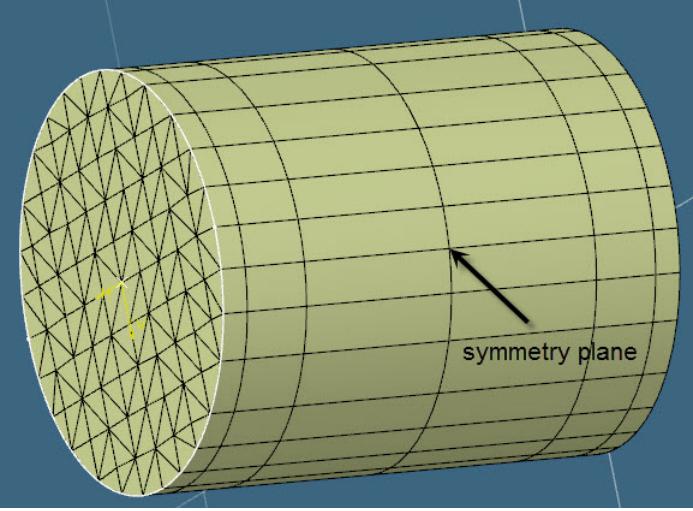
default value: 1**RatioValue**

type: double valued to the size ratio between the first and last layer thicknesses.	This attribute specifies the size ratio R between the first layer thickness $h1$ and last layer thickness $h2$. $R = h2 / h1$ WARNING: It is taken into account only if the Type attribute is equal to 2 or 3.	
--	---	--

NbElementsValue

type: integer valued to the expected number of layers.	This attribute specifies the number of layers to be added to the volume mesh.
---	---

Symmetry

type: boolean valued to create an extrusion which is symmetric.	Attribute which specifies if we want to create a symmetric extrusion, the symmetrical plane is located at the middle of the extrusion. WARNING: It is taken into account only if the Type attribute is equal to 2 or 3.	
--	--	--

AutomaticMeshCapture

type: integer valued to:	Attribute which specifies if external meshes must be captured. This attribute is linked to the attribute AutomaticMeshCaptureTolValue .
---------------------------------	--

AutomaticMeshCaptureTolValue

type: double valued to the maximum distance under	Attribute which specifies the maximum distance for mesh capture: if a mesh (element edges) is farthest from the support than this tolerance, it will not be captured even if the AutomaticMeshCapture attribute is valued to 2. WARNING: to precise exactly the list of mesh parts to capture, the method SetMeshPartsToCapture of
--	--

which mesh is captured.

default value: 0.

SimMeshPart object can be used:

- SetMeshPartsToCapture ()

To retrieve the parents mesh parts another method of **SimMeshPart** object can be used:

- Parent ()

History

Version: 1 [November 2014] Document created

Tetrahedron Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Tetrahedron Mesh Part** attributes and its local specifications.

[Tetrahedron Mesh Part Creation](#)

[Tetrahedron Mesh Part Support Definition](#)

[Global Mesh Specifications](#)

- [MeshSizeValue](#)
- [AutoMap](#)
- [ElementOrder](#)
- [AbsoluteSag](#)
- [AbsoluteSagValue](#)
- [BoundaryLayersMode](#)
- [BoundaryLayersDistribution](#)
- [BoundaryLayersNumber](#)
- [BoundaryLayersRatio](#)
- [BoundaryLayersHeight1](#)
- [BoundaryLayersTetsOnly](#)
- [ProportionalSag](#)
- [ProportionalSagValue](#)
- [MinMeshSize](#)
- [MinCellsPerGap](#)
- [MinCellsPerGapValue](#)
- [GrowthRatioSurface](#)
- [GrowthRatioVolume](#)

[Global Topology Specifications](#)

- [AngleBetweenCurves](#)
- [AngleBetweenFaces](#)
- [CurvatureAngle](#)
- [GeometrySimplification](#)
- [GeometrySimplificationValue](#)
- [LogosSuppression](#)
- [LogosMaxHeight](#)
- [LogosMaxSize](#)

[Local Specifications](#)

[Local Topology Specifications](#)

- [CATFmtExternalPointLocalSpec](#) (Imposed Points)
- [CATFmtPreserveGeometries](#) (Preserved Boundaries)

[Local Mesh 2D Specifications](#)

- [CATFmtLocalSpecLocalSize](#) (Local Mesh Size)

[Local Mesh 3D Specifications](#)

- [CATFmtBoundaryLayerSpecification](#) (Boundary Layers)
- [CATFmtExcludedBoundaryLayerSpecification](#) (Excluded Boundary Layers)

Tetrahedron Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Surface Quadrangle Mesh Part, the late type is "CATFmt3DRulesMesher".

The Tetrahedron Mesh Part can be created by using **Add** method of **SimMeshParts** object.

Tetrahedron Mesh Part Support Definition

The Tetrahedron Mesh Part support can be initialized and managed using the **AddSupport** method of **SimMeshPart** object:

- **AddSupport** (*iSupport As AnyObject*) to add a new support to the mesh part

Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods:

- **SetAttributeValue** (*iSetType As String, iAttribute As String, iValue*) to set a mesh attribute with an integer value
- **GetAttributeValue** (*iSetType As String, iAttribute As String, oValue*) to get a mesh attribute with an integer value

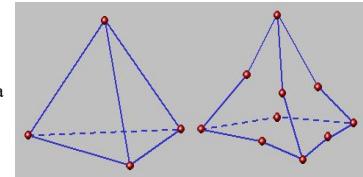
iType is the type of specification and **iType** = "Mesh" for Global Mesh Specifications.

ElementOrder

type: integer valued to:

- 1: to get a mesh with linear tetrahedra (*TE4*)
- 2: to get a mesh with quadratic tetrahedra (*TE10*)

Attribute which specifies the number of intermediate nodes per element edge, as shown on the pictures (the left one shows a linear tetrahedron and the right one shows a parabolic tetrahedron).



default value: 1

MeshSizeValue

type: double valued to mesh elements size.

Attribute which specifies the mesh element target size that mesher tries to respect on the surface.

default value: 10

AutoMap

type: integer valued to:

- 1: do not map mesh the surfaces
- 2: map mesh the surfaces where appropriate

default value: 2

This attribute specifies whether map mesh is applied on surfaces where appropriate.

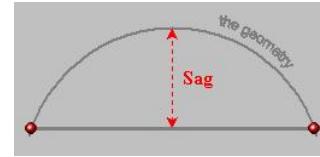
AbsoluteSag

type: integer valued to:

- 1: to create a mesh that does not respect sag
- 2: to create a mesh that respects sag

Attribute which specifies if the mesh is created with a maximum imposed sag.

default value: 1



AbsoluteSagValue

type: double valued to the target maximum sag when **AbsoluteSag** attribute is valued to 2.

Attribute which specifies the maximum value of the absolute sag (distance between the geometry of the mesh).

default value: 0

WARNING: It is taken into account only if the **AbsoluteSag** attribute is equal to 2.

BoundaryLayersMode (only available starting from 3DEXPERIENCE 2015x FD01)

type: integer valued to:

- 1: to get a mesh without boundary layers
- 2: to get a mesh with boundary layers

This attribute specifies whether boundary layers should be added everywhere in the volume mesh or not.

default value: 1

BoundaryLayersDistribution (only available starting from 3DEXPERIENCE 2015x FD01)

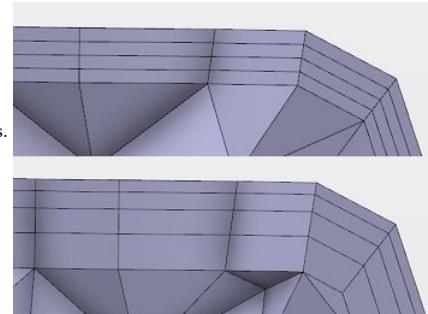
type: integer valued to:

- 1: to get an uniform distribution
- 2: to get an arithmetic distribution
- 3: to get a geometric distribution

This attribute specifies the size distribution to be applied to the boundary layers.

The picture shows how a uniform distribution (at the top) differs from a geometric one (at the bottom, using a ratio of 3).

default value: 1



BoundaryLayersNumber (only available starting from 3DEXPERIENCE 2015x FD01)

type: integer valued to the expected number of boundary layers.

This attribute specifies the number of boundary layers to be added to the volume mesh.

default value: 1

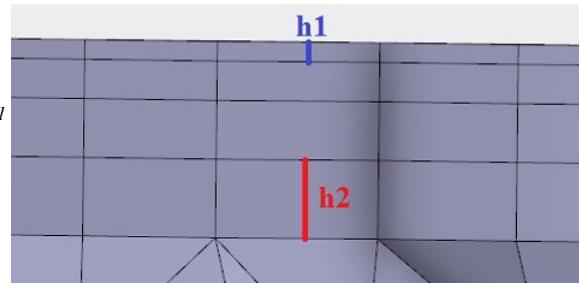
BoundaryLayersRatio (only available starting from 3DEXPERIENCE 2015x FD01)

type: *double* valued to the size ratio between the first and last layer thicknesses.

default value: 1.0

This attribute specifies the size ratio R between the first layer thickness $h1$ and last layer thickness $h2$. $R = h2 / h1$

WARNING: It is taken into account only if the **BoundaryLayersDistribution** attribute is equal to 2 or 3.



BoundaryLayersHeight1 (only available starting from 3DEXPERIENCE 2015x FD01)

type: *double* valued to the expected thickness of the first layer of boundary elements.

default value: 1.0

This attribute specifies the thickness of the first layer of elements in the boundary layers.

BoundaryLayersTetsOnly (only available starting from 3DEXPERIENCE 2015x FD01)

type: *integer* valued to:

- 1: to get wedges and pyramids in the boundary layers
- 2: to get tetrahedra only in the boundary layers

This attribute specifies the elements to be generated in the boundary layers. If set to 2, elements in the boundary layers will be split into tetrahedra.

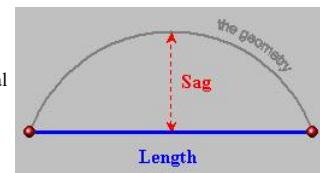
default value: 1

ProportionalSag

type: *integer* valued to:

- 1: to create a mesh that does not respect proportional sag
- 2: to create a mesh that respects proportional sag

Attribute which specifies if the mesh is created with a imposed maximum proportional sag which is the ratio of the Sag by the Length of a mesh edge element (*Sag/Length*).



default value: 1

ProportionalSagValue

type: *double* valued to target proportional sag when ProportionalSag attribute is valued to 2.

Attribute which specifies the maximum value of the proportional sag.

default value: 0

WARNING: It is taken into account only if the **ProportionalSag** attribute is equal to 2.

MinMeshSize

type: *double* valued to the minimum mesh elements size. Attribute which specifies the minimum mesh element size that mesher tries to respect on the surface.

default value: 0.2 * MeshSizeValue

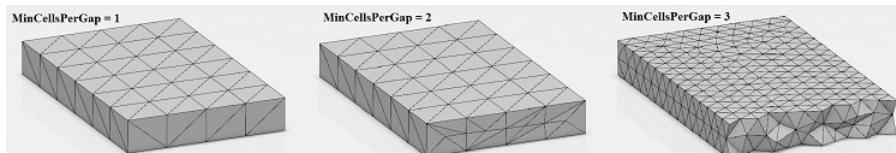
This attribute is only needed when **AbsoluteSag** is set to 2 or **ProportionalSag** is set to 2.

MinCellsPerGap

type: *integer* valued to:

Attribute which specifies the treatment to be applied in narrow regions. Tetrahedra can either be split to ensure two layers of elements everywhere or the surface mesh can be refined, as shown on the picture below.

- 1: to keep the default mesh
- 2: to split tetrahedra
- 3: to refine the surface



default value: 1

MinCellsPerGapValue

type: *integer* valued to the expected number of layers across the thickness. Attribute which specifies the number of expected layers across the thickness. The surface mesh is refined (see picture above).

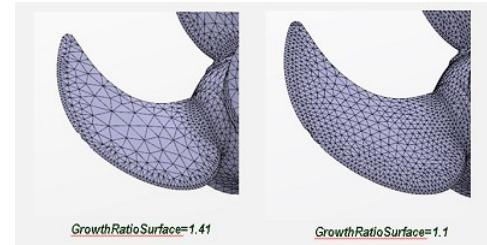
default value: 0

WARNING: It is taken into account only if the **MinCellsPerGap** attribute is equal to 3.

GrowthRatioSurface

type: *double* valued to the mesh size growth. Attribute which controls the mesh size growth between neighboring elements on the surface.

default value: 1.41 **WARNING:** It is taken into account only if the **AbsoluteSag**, **ProportionalSag** or **MinCellsPerGap** attribute is equal to 2.

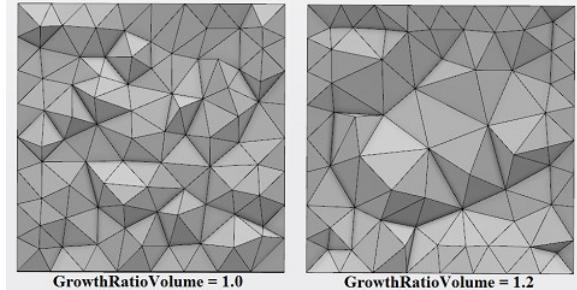


GrowthRatioVolume

type: *double* valued to the expected growth ratio throughout the volume.

default value: 1.0

This attribute specifies the amount by which adjacent elements through the volume can differ in size. It controls the smoothness of transitions in mesh size, as shown on the pictures on the right.



Global Topology Specifications

Global topology specifications can be set or retrieved using following methods:

- **SetAttributeValue** (*iSetType As String, iAttribute As String, iValue*) to set a topology attribute
- **GetAttributeValue** (*iSetType As String, iAttribute As String, oValue*) to get a topology attribute

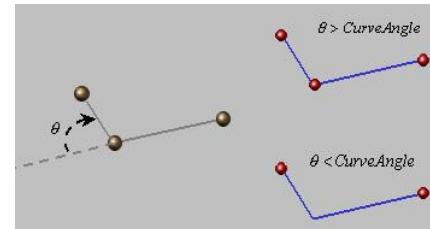
iType is the type of specification and **iType** = "Topology" for Global Topology Specifications.

AngleBetweenCurves

type: *double (radian unit)* valued to the angle under which vertices are inactivated

default value: 20°

Attribute which specifies the maximum angle under which a vertex connected exactly to two constrained edges can be inactivated.

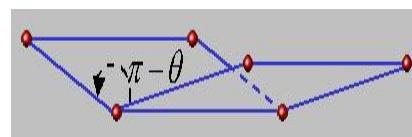


AngleBetweenFaces

type: *double (radian unit)* valued to the angle under which edges are inactivated.

default value: 20°

Attribute which specifies the maximum angle under which an edge connected exactly to two faces can be inactivated during topology computation.

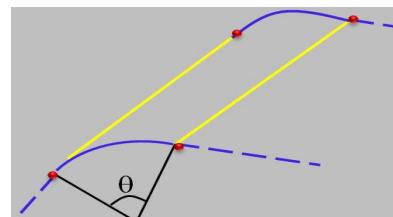


CurvatureAngle

type: *double (radian unit)* valued to the angle over which edges are activated.

default value: 20°

Attribute which specifies the minimum angle computed along filleted surfaces over which edges are activated. For a given fillet, when the computed curvature angle is greater than **CurvatureAngle** all boundary edges of this fillet are activated otherwise they are not.



GeometrySimplification

type: *integer* valued to:

- 1: to keep grouping topology without additional simplification
- 2: to compute topology simplification

default value: 2

Attribute which specifies if topology simplification is computed. Topology simplification objective is to minimize the number of bad elements regarding the Min Height element quality criterion.

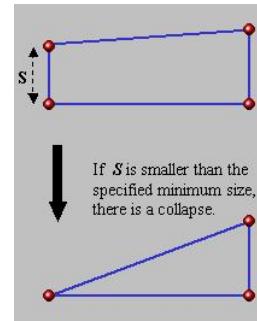
GeometrySimplificationValue

type: `double` valued to the Min Height criterion targeted for mesh elements.

default value: `0.1 * MeshSizeValue`

When topology simplification is computed, different topology modifications are done automatically using a set of topological operators in order to respect Min Height element quality criterion:

- Vertex Creation/Activation/Inactivation
- Edge Creation/Activation/Inactivation
- Edge collapse (see picture)
- Edge Merge

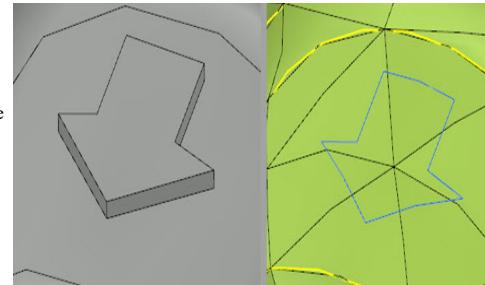


LogosSuppression

type: `integer` valued to: Attribute which specifies whether logos suppression will be done.

- 1: to ignore logos control suppression
- 2: to control logos suppression

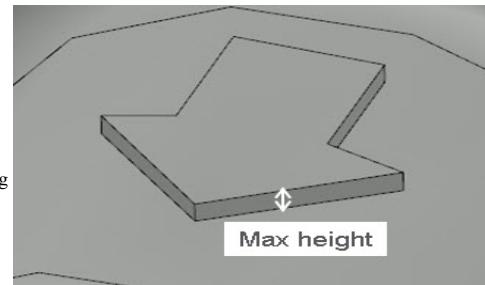
default value: 1 For example on the right picture, an ignored logo can be seen.



LogosMaxHeight

type: `double` valued to the max height of a detectable logo Attribute which specifies the maximal height allowed to detect logos.

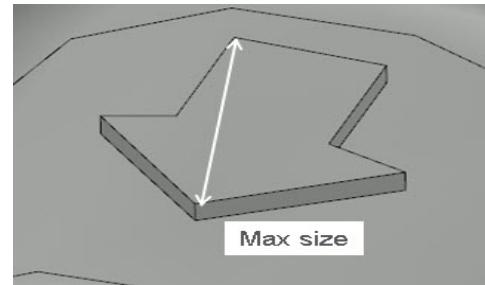
default value: `0.1 * MeshSizeValue` The maximum height of a logo can be defined as the maximum distance of any points on the logo faces to the extrapolated bounding surfaces.



LogosMaxSize

type: `double` valued to the max size of a detectable logo Attribute which specifies the maximal size allowed to detect logos.

default value: `MeshSizeValue` The maximum size of a logo can be defined as the maximum distance between two points on the logo boundary (maximum diameter).



Local Specifications

The Tetrahedron Mesh Part has two kind of local specifications: **topology** and **mesh** local specifications. These specifications are managed by collection object:

- **SimMeshSpecifications:** for the collection of local mesh specifications
- **SimTopologySpecifications:** for the collection of local topology specifications

You should use the following methods of **SimMeshPart** object to retrieve these collections:

- **GetMeshSpecifications** to retrieve the local mesh specification collection
- **GetTopologySpecification** to retrieve the local topology specification collection

Local specifications are managed from the collection objects:

- **Add (iType As String)** to create a new local specification with:
 - iType is the late type of the specification to create
- **Item (iIndex)** to retrieve a local specification with:
 - iIndex is the index of the local specification in its collection
- **Remove (iIndex)** to delete a local specification with:
 - iIndex is the index of the local specification in its collection

Attribute of local specification are set using following method from **SimMeshSpecification** or **SimTopologySpecification** object depending of the kind of local specification:

- **SetAttributeValue** (*iAttribute As String, iValue*) to set a local specification attribute value

In a manner similar, attribute of local specification are retrieved using following method:

- **GetAttributeValue** (*iAttribute As String, iValue*) to retrieve a local specification

Geometric attribute of local specification (supports) are managed using **SimLinkAccess** object:

- **AddLink** (*iName As String, iLink As AnyObject*)
- **RemoveAllLinks** (*iName As String*)
- **RemoveLink** (*iName As String, iLink As AnyObject*)
 - *iName* is the name of the link container, here it's "**ConnectorList**"

Local Topology Specifications

CATFmtExternalPointLocalSpec

Specification attributes are managed using **SimTopologySpecification** object:

- Geometric attribute (supports)

Other attributes are managed using (GetAttribute, SetAttribute):

- **ToleranceValue**: *double* valuated to the maximum distance between external point and its projection on geometry to mesh.

default value: 0.

- **ProjectOnGeometry**: *integer* valuated to the target geometry choice

default value: 1

Description: This specification is used in order to project **external** points (points that don't belong to the geometry to mesh) in order to constrain the mesh. External points definition (supports) and projection tolerance are mandatory. The overall behavior is the following:

- if the **distance** between an external point and the geometry to mesh is **greater** than the **ToleranceValue**, the **point will not be projected**
- if the **distance** between an external point and the geometry to mesh is **lower** than the **ToleranceValue**, the **point will be projected**. The final location of this point depends on **ProjectOnGeometry** attribute:
 - if **ProjectOnGeometry** is equal to **1**, the **node** representing the point will be **located** exactly on the **projection point on the surface geometry**
 - if **ProjectOnGeometry** is equal to **2**, the **node** representing the point projection will be **located** exactly on the **external point (same Cartesian coordinates)**

CATFmtPreserveGeometries

Specification attributes are managed using **SimTopologySpecification** object.

- Only geometric attribute (supports)

Description: This specification is used in order to preserve a set of faces, edges or points while meshing i.e. to prevent them from being simplified.

The picture shows how a small geometrical element can be preserved by selecting its faces as a support of the preserved geometries specification.



Local Mesh 2D Specifications

CATFmtLocalSpecLocalSize

Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

Other attributes are managed using (GetAttribute, SetAttribute):

- **SizeValue**: *double* valuated to the target size on the support
- **AbsoluteSag**: *int* valuated to
 - 1: to specify only a target mesh size (default)
 - 2: to specify a local sag as well
- **AbsoluteSagValue**: *double* valuated to the target sag on the support (only taken into account if the **AbsoluteSag** attribute above is set to 2)

Description: This specification is used in order to specify a local mesh size or sag on a set of faces.

Local Mesh 3D Specifications

CATFmtBoundaryLayerSpecification

Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

Other attributes are managed using (GetAttribute, SetAttribute):

- **Offset**: *double* valuated to the total thickness of the boundary layers
- **TetsOnly**: *int* valuated to
 - 1: to get wedges and pyramids in the boundary layers (default)
 - 2: to get tetrahedra only in the boundary layers
- **Type**: *int* valuated to
 - 1: to get an uniform distribution (default)
 - 2: to get an arithmetic distribution

Description: This specification is used to create boundary layers only on a given set of faces.

- 3: to get a geometric distribution
- **NbElementsValue:** int valued to the number of expected boundary layers
- **RatioValue:** double valued to the expected ratio of thicknesses between the first and last boundary layers
- **Symmetry:** int valued to
 - 1: to get a standard distribution (default)
 - 2: to get a symmetric distribution

It can be used to override the global specification locally.

The **RatioValue** and **Symmetry** attributes are only taken into account if **Type** is set to 2 or 3.

CATFmtExcludedBoundaryLayerSpecification (only available starting from 3DEXPERIENCE 2015x FD01)

Specification attributes are managed using **SimMeshSpecification** object.

- Only geometric attribute (supports)

Description: This specification is used to prevent the creation of boundary layers on its support faces.

This is particularly useful when boundary layers are needed everywhere (they can be applied through Global Mesh Specifications) except in a handful of locations.

History

Version: 1 [October 2014] Document created

Tetrahedron Filler Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Tetrahedron Filler Mesh Part** attributes and its local specifications.

[Tetrahedron Filler Mesh Part Creation](#)

[Tetrahedron Filler Mesh Part Support Definition](#)

[Global Mesh Specifications](#)

- [ElementOrder](#)
- [MinCellsPerGap](#)
- [Propagation](#)

[Local Specifications](#)

[Local Mesh Specifications:](#)

- [CATFmtBoundaryLayerSpecification](#) (Boundary Layers)

Tetrahedron Filler Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the tetrahedron filler mesh part, the late type is "CATFmtGHS3DMesher".

The tetrahedron filler mesh part can be created by using **Add** method of **SimMeshParts** object.

Tetrahedron Filler Mesh Part Support Definition

Tetrahedron Filler Mesh Part support can be initialized and managed using the **AddSupport** method of **SimMeshPart** object:

- **AddSupport** (*iSupport As AnyObject*) to add a new support to the mesh part

Global Mesh Specifications

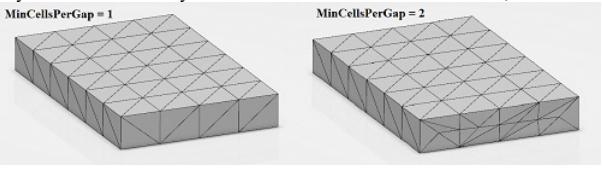
Global mesh specifications can be set or retrieved using following methods of the **SimMeshPart** object:

- **SetAttributeValue** (*iSetType As String*, *iAttribute As String*, *iValue*) to set a mesh attribute where:
 - *iSetType*=**"Mesh"**
 - *iAttribute* is the attribute's name
 - *iValue* is the attribute's value
- **GetAttributeValue** (*iSetType As String*, *iAttribute As String*, *oValue*) to get a mesh attribute where:
 - *iSetType*=**"Mesh"**
 - *iAttribute* is the attribute's name
 - *oValue* is the attribute's value

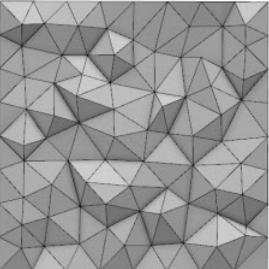
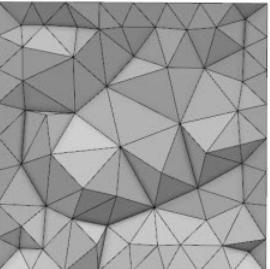
ElementOrder

<p>type: integer valued to:</p> <ul style="list-style-type: none"> • 1: to get a mesh with linear tetrahedra (<i>TE4</i>) • 2: to get a mesh with quadratic tetrahedra (<i>TE10</i>) <p>default value: 1</p>	<p>Attribute which specifies the number of intermediate nodes per element edge, as shown on the pictures (the left one shows a linear tetrahedron and the right one shows a parabolic tetrahedron).</p> 
---	--

MinCellsPerGap

<p>type: <i>integer</i> valued to:</p> <ul style="list-style-type: none"> • 1: to keep the default mesh • 2: to split tetrahedra <p>default value: 1</p>	<p>Attribute which specifies the treatment to be applied in narrow regions. Tetrahedra can either be split to ensure two layers of elements everywhere or the surface mesh can be refined, as shown on the picture below.</p> 
--	--

Propagation

<p>type: <i>double</i> valued to the expected growth ratio throughout the volume.</p> <p>default value: 1.0</p>	<p>This attribute specifies the amount by which adjacent elements through the volume can differ in size. It controls the smoothness of transitions in mesh size, as shown on the pictures on the right.</p>	 <i>Propagation = 1.0</i>	 <i>Propagation = 1.2</i>
---	---	--	---

Local Specifications

The local specifications are managed by collection object:

- **SimTopologySpecifications** is the collection of **SimTopologySpecification** object (local topology specification)
- **SimMeshSpecifications** is the collection of **SimMeshSpecification** object (local mesh specification)

You should use the following methods of **SimMeshPart** object to retrieve this collection:

- **GetTopologySpecification** to retrieve the local topology specification collection
- **GetMeshSpecification** to retrieve the local mesh specification collection

Local specifications are managed from the collection object:

- **Add** (*iType As String*, *oSpec*) to create a new local specification with:
 - *iType* is the late type of the specification to create
 - *oSpec* is a local specification (**SimTopologySpecification** or **SimMeshSpecification**)
- **Item** (*iIndex*, *oSpec*) to retrieve a local specification with:
 - *iIndex* is the index of the local specification in its collection
 - *oSpec* is a local specification (**SimTopologySpecification** or **SimMeshSpecification**)
- **Remove** (*iIndex*) to delete a local specification with:
 - *iIndex* is the index of the local specification in its collection

Local specification attributes are managed using the local specification object:

- **SetAttributeValue** (*iAttribute As String*, *iValue*) to set a local specification attribute where:
 - *iAttribute* is the attribute's name
 - *iValue* is the attribute's value
- **GetAttributeValue** (*iAttribute As String*, *oValue*) to get a local specification attribute where:
 - *iAttribute* is the attribute's name
 - *oValue* is the attribute's value

Geometric attribute of local specification (supports) are managed using **SimLinkAccess** object:

- **AddLink** (*iName As String*, *iLink As AnyObject*)
- **RemoveAllLinks** (*iName As String*)
- **RemoveLink** (*iName As String*, *iLink As AnyObject*)
 - *iName* is the name of the link container, here it's "**ConnectorList**"

Local Mesh Specifications

CATFmtBoundaryLayerSpecification

Specification attributes are managed using **SimMeshSpecification** object:

- Geometric attribute (supports)

Other attributes are managed using (GetAttribute, SetAttribute):

- **Offset:** *double* valued to the total thickness of the boundary layers
- **TetsOnly:** *int* valued to
 - 1: to get wedges and pyramids in the boundary layers (default)
 - 2: to get tetrahedra only in the boundary layers
- **Type:** *int* valued to
 - 1: to get an uniform distribution (default)
 - 2: to get an arithmetic distribution
 - 3: to get a geometric distribution
- **NbElementsValue:** *int* valued to the number of expected boundary layers
- **RatioValue:** *double* valued to the expected ratio of thicknesses between the first and last boundary layers

Description: This specification is used to create boundary layers only on a given set of faces.

It can be used to override the global specification locally.

- **Symmetry:** int valued to
 - 1: to get a standard distribution (default)
 - 2: to get a symmetric distribution

The **RatioValue** and **Symmetry** attributes are only taken into account if **Type** is set to **2** or **3**.

History

Version: 1 [October 2014] Document created

Translation Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Translation Mesh Part** attributes and its local specifications.

Translation Mesh Part Creation

Translation Mesh Part Support Definition:

- [Geometry](#)

Global Mesh Specifications:

- [Distance](#)
- [AutomaticMeshCapture](#)
- [AutomaticMeshCaptureTolValue](#)
- [NbCopies](#)

Translation Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Translation mesh part, the late type is "CATFmTranslationMesher"

The Translation mesh part can be created by using **Add** method of **SimMeshParts** object.

Translation Mesh Part Support Definition

Translation Mesh Part support can be set using following method of the **SimMeshPart** object:

- **AddSupport** (*iSupport As AnyObject*) to add a new support to the mesh part

Geometry

The Translation Mesh Part direction can be set or unset using following methods of the **SimMeshPart** object:

- **SetExternalReference** (*iSetType As String, iAttribute As String, iSupport As AnyObject*) to set the translation direction where:
 - *iSetType*=**"Topology"**
 - *iAttribute*=**"Geometry"**
 - *iSupport* is a geometric line
- **RemoveExternalReference** (*iSetType As String, iAttribute As String, iSupport As AnyObject*) to unset the translation direction where:
 - *iSetType*=**"Topology"**
 - *iAttribute*=**"Geometry"**
 - *iSupport* is a geometric line

Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods of the **SimMeshPart** object:

- **SetAttributeValue** (*iSetType As String, iAttribute As String, iValue*) to set a mesh attribute where:
 - *iSetType*=**"Mesh"**
 - *iAttribute* is the attribute's name
 - *iValue* is the attribute's value
- **GetAttributeValue** (*iSetType As String, iAttribute As String, oValue*) to get a mesh attribute where:
 - *iSetType*=**"Mesh"**
 - *iAttribute* is the attribute's name
 - *oValue* is the attribute's value

Distance

type: <i>double</i> valued to a length. default value: 0.2	Attribute which specifies the distance between the original mesh and the new mesh that we want to create thanks to a translation transformation.
---	--

AutomaticMeshCapture

type: <i>integer</i> valued to: • 1: to ignore automatic mesh capture	Attribute which specifies if external meshes must be captured. This attribute is linked to the attribute
---	--

<ul style="list-style-type: none"> • 2: to compute automatic mesh capture <p>default value: 1</p>	AutomaticMeshCaptureTolValue.
---	--------------------------------------

AutomaticMeshCaptureTolValue

<p>type: <i>double</i> valued to the maximum distance under which mesh is captured.</p> <p>default value: 0.</p>	<p>Attribute which specifies the maximum distance for mesh capture: if a mesh (element edges) is farthest from the support than this tolerance, it will not be captured even if the AutomaticMeshCapture attribute is valued to 2.</p> <p>WARNING: to precise exactly the list of mesh parts to capture, the method SetMeshPartsToCapture of SimMeshPart object can be used:</p> <ul style="list-style-type: none"> • SetMeshPartsToCapture () <p>To retrieve the parents mesh parts another method of SimMeshPart object can be used:</p> <ul style="list-style-type: none"> • Parent ()
--	--

NbCopies

<p>type: <i>integer</i> valued to the number of translated meshes.</p> <p>default value: 1</p>	<p>Attribute which specifies the number of translated meshes that we want to create.</p>
--	--

History

Version: 1 [October 2014] Document created

Translation Extrusion Mesh Part Attributes

Technical Article

Abstract

This article presents an overview of the **Translation Extrusion Mesh Part** attributes and its local specifications.

[Translation Extrusion Mesh Part Creation](#)[Translation Extrusion Mesh Part Support Definition:](#)

- [Geometry](#)

[Global Mesh Specifications:](#)

- [Start](#)
- [End](#)
- [Type](#)
- [RatioValue](#)
- [NbElementsValue](#)
- [Symmetry](#)
- [AutomaticMeshCapture](#)
- [AutomaticMeshCaptureTolValue](#)

Translation Extrusion Mesh Part Creation

A "late type" must be given to indicate which type of mesh part is to be created.

In the case of the Translation Extrusion mesh part, the late type is "CATFmtExtrusionTranslationMesher".

The Translation Extrusion mesh part can be created by using **Add** method of **SimMeshParts** object.

Translation Extrusion Mesh Part Support Definition

Translation Extrusion Mesh Part support can be set using following method of the **SimMeshPart** object:

- **AddSupport (iSupport As AnyObject)** to add a new support to the mesh part

Geometry

The Translation Extrusion Mesh Part direction can be set or unset using following methods of the **SimMeshPart** object:

- **SetExternalReference (iSetType As String, iAttribute As String, iSupport As AnyObject)** to set the translation direction where:
 - **iSetType="Topology"**
 - **iAttribute="Geometry"**
 - **iSupport** is a geometric line
- **RemoveExternalReference (iSetType As String, iAttribute As String, iSupport As AnyObject)** to unset the translation direction where:
 - **iSetType="Topology"**
 - **iAttribute="Geometry"**
 - **iSupport** is a geometric line

Global Mesh Specifications

Global mesh specifications can be set or retrieved using following methods of the **SimMeshPart** object:

- **SetAttributeValue** (*iSetType As String, iAttribute As String, iValue*) to set a mesh attribute where:
 - *iSetType*="Mesh"
 - *iAttribute* is the attribute's name
 - *iValue* is the attribute's value
- **GetAttributeValue** (*iSetType As String, iAttribute As String, oValue*) to get a mesh attribute where:
 - *iSetType*="Mesh"
 - *iAttribute* is the attribute's name
 - *oValue* is the attribute's value

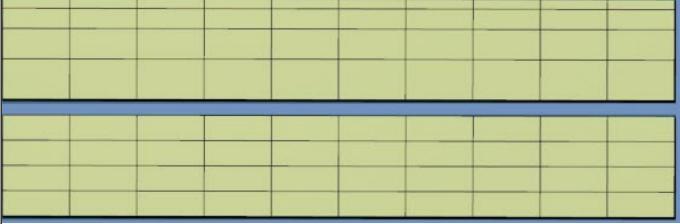
Start

type: <i>double</i> valued to a length. default value: 0.	Attribute which specifies the distance from the original mesh where the extrusion will begin. This attribute is linked to the attribute End .
--	--

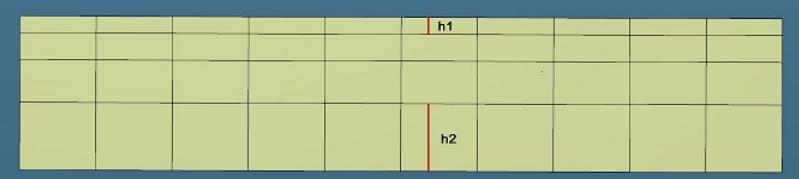
End

type: <i>double</i> valued to a length. default value: 0.	Attribute which specifies the distance from the original mesh where the extrusion will stop. This attribute is linked to the attribute Start .
--	---

Type

type: <i>integer</i> valued to: <ul style="list-style-type: none">• 1: to get an uniform distribution• 2: to get an arithmetic distribution• 3: to get a geometric distribution default value: 1	This attribute specifies the size distribution to be applied to the mesh. The picture shows how a uniform distribution (at the bottom) differs from a geometric one (at the top, using a ratio of 3).	
--	--	---

RatioValue

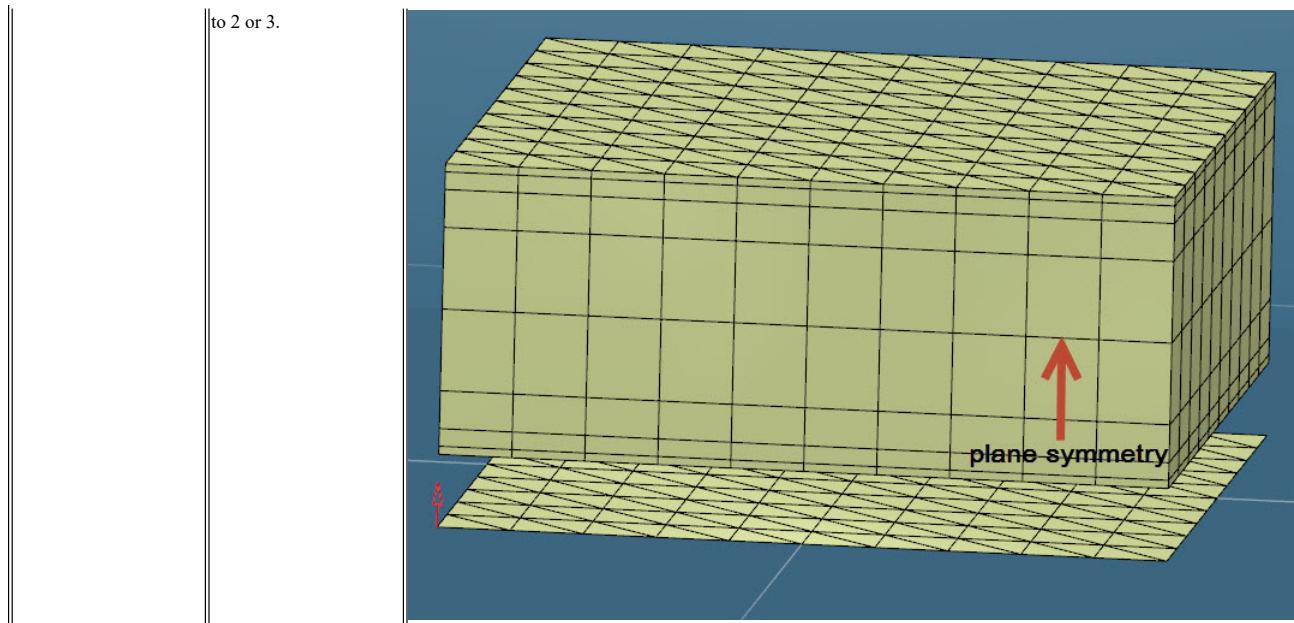
type: <i>double</i> valued to the size ratio between the first and last layer thicknesses. default value: 1.0	This attribute specifies the size ratio <i>R</i> between the first layer thickness <i>h1</i> and last layer thickness <i>h2</i> . $R = h2 / h1$ WARNING: It is taken into account only if the Type attribute is equal to 2 or 3.	
--	--	--

NbElementsValue

type: <i>integer</i> valued to the expected number of layers. default value: 1	This attribute specifies the number of layers to be added to the volume mesh.
---	---

Symmetry

type: <i>boolean</i> valued to create an extrusion which is symmetric. default value: false	Attribute which specifies if we want to create a symmetric extrusion, the symmetrical plane is located at the middle of the extrusion. WARNING: It is taken into account only if the Type attribute is equal
--	---



AutomaticMeshCapture

type: <i>integer</i> valued to:	
<ul style="list-style-type: none"> • 1: to ignore automatic mesh capture • 2: to compute automatic mesh capture 	Attribute which specifies if external meshes must be captured. This attribute is linked to the attribute AutomaticMeshCaptureTolValue .
default value: 1	

AutomaticMeshCaptureTolValue

type: <i>double</i> valued to the maximum distance under which mesh is captured.	Attribute which specifies the maximum distance for mesh capture: if a mesh (element edges) is farthest from the support than this tolerance, it will not be captured even if the AutomaticMeshCapture attribute is valued to 2.
default value: 0.	<p>WARNING: to precise exactly the list of mesh parts to capture, the method SetMeshPartsToCapture of SimMeshPart object can be used:</p> <ul style="list-style-type: none"> • SetMeshPartsToCapture () <p>To retrieve the parents mesh parts another method of SimMeshPart object can be used:</p> <ul style="list-style-type: none"> • Parent ()

History

Version: 1 [November 2014] Document created

Groups and Their Attributes

Technical Article

Abstract

The following reference article describes the different types of groups, their attributes and usage.

This paper describes how to access to:

- [Group Creation](#)
- [Group Type Definition](#)

This paper explains attributes of seven type of groups:

- The Geometric Group. See [Geometric Group Attributes](#). *The description of all attributes of the Geometry Group.*
- The Proximity Group. See [Proximity Group Attributes](#). *The description of all attributes of the Proximity Group.*
- The Spatial Group. See [Spatial Group Attributes](#). *The description of all attributes of the Spatial Group.*
- The Boundary Group. See [Boundary Group Attributes](#). *The description of all attributes of the Boundary Group.*
- The Union Group. See [Union Group Attributes](#). *The description of all attributes of the Union Group.*
- The Intersection Group. See [Intersection Group Attributes](#). *The description of all attributes of the Intersection Group.*
- The Difference Group. See [Difference Group Attributes](#). *The description of all attributes of the Difference Group.*

Group creation

Group creation can be managed the `SimGroupSet` object by calling the `Add` method from it.

A "late type" must be given to indicate which type of group is to be created:

Group

Late type

Geometric Group	CATFmtGroupByAssociativity
Proximity Group	CATFmtGroupByProximity
Spatial Group	CATFmtGroupSpatial
Boundary Group	CATFmtGroupByBoundary
Union Group	CATFmtGroupUnion
Intersection Group	CATFmtGroupIntersection
Difference Group	CATFmtGroupDifference

Group Type

Group type can be set by the `SetGroupType` method of the `SimGroup` object.

A string must be given to indicate which kind of elements the group will contain:

String	Description
CATFmtGroupTypeNode	The group contains only the nodes of the support
CATFmtGroupTypeEdge	The group contains only edges
CATFmtGroupTypeFace	The group contains only 3D elements faces of the support
CATFmtGroupTypeElement	The group contains only elements of the support

Group Support

The support of the groups is managed by the `SimLinkAccess` object:

- `AddLink (iName As String, iLink As AnyObject)`
- `RemoveAllLinks (iName As String)`
- `RemoveLink (iName As String, iLink As AnyObject)`
 - `iName` is the name of the link container, here it's :
 - "MainSupport" for all the groups which need a support.
 - "Boundary" to specify boundary geometry of boundary group.
 - "MainSupport" for the Groups A support list of difference group.
 - "DifferenceGroups" for the Groups B support list of difference group.

Geometry Group Attributes

`CnxElementAllowed` (type `integer`): valued to:

- 1: to don't allow connection element in the group content
- 2: to allow connection element in the group content

Attribute which specifies if connection elements are allowed in the group content.

Proximity Group Attributes

`Tolerance` (type `double`): valued to group tolerance. Attribute which specifies the tolerance in which the mesh entities will be captured.

`CnxElementAllowed` (type `integer`): valued to:

- 1: to don't allow connection element in the group content
- 2: to allow connection element in the group content

Attribute which specifies if connection elements are allowed in the group content.

Spatial Group Attributes

`Mode` (type `integer`): valued to:

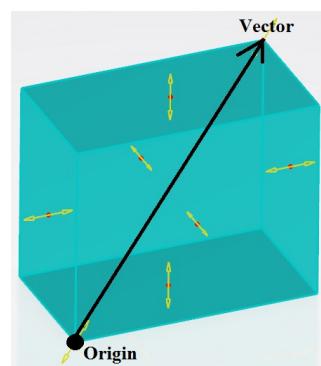
- 1: to get a box mode
- 2: to get a sphere mode

Attribute which specifies the kind of the bounding selection

The box is defined by its lower left corner and a vector transforming it into the upper right corner, as shown on the picture below.

The following attributes are used to define the box in case of spatial group with a box mode.

- `OriginX` (type `double`): the X coordinate of the lower left corner of the box
- `OriginY` (type `double`): the Y coordinate of the lower left corner of the box
- `OriginZ` (type `double`): the Z coordinate of the lower left corner of the box
- `Vector0X` (type `double`): the dimension of the box along the X axis
- `Vector1Y` (type `double`): the dimension of the box along the Y axis
- `Vector2Z` (type `double`): the dimension of the box along the Z axis



The following attributes are used to define the sphere in case of spatial group with a sphere mode.

- `CenterX` (type `double`): the X coordinate of the sphere center
- `CenterY` (type `double`): the Y coordinate of the sphere center
- `CenterZ` (type `double`): the Z coordinate of the sphere center
- `Radius` (type `double`): the radius of the sphere

Attributes which specifies the sphere

`CnxElementAllowed` (type `integer`): valued to:

Attribute which specifies if connection elements are allowed in the group content.

- 1: to don't allow connection element in the group content

- 2: to allow connection element in the group content

Boundary Group Attributes

No specific attribute for this group.

Union Group Attributes

CnxElementAllowed (type integer): valued to:

- 1: to don't allow connection element in the group content Attribute which specifies if connection elements are allowed in the group content.
- 2: to allow connection element in the group content

Intersection Group Attributes

CnxElementAllowed (type integer): valued to:

- 1: to don't allow connection element in the group content Attribute which specifies if connection elements are allowed in the group content.
- 2: to allow connection element in the group content

Difference Group Attributes

CnxElementAllowed (type integer): valued to:

- 1: to don't allow connection element in the group content Attribute which specifies if connection elements are allowed in the group content.
- 2: to allow connection element in the group content

History

Version: 1 [Oct 2014] Document created

Managing FEM group features

This use case primarily focuses on the methodology to create/delete FEM groups.

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdfeaProductWithoutFEM.3dxml` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaModeling\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaGroupsSource.htm](#)

This use case can be divided in eight steps:

1. [Searches and opens model which is named as "CAAScdfeaProductWithoutFEM"](#)
2. [Retrieves the root product](#)
3. [Retrieves the PartBody](#)
4. [Creates the FEM representation and retrieve its root object](#)
5. [Creates/Retrieves the group set](#)
6. [Creates a geometric group](#)
7. [Creates a proximity group](#)
8. [Remove the proximity group](#)

1. **Searches and opens model which is named as "CAAScdfeaProductWithoutFEM"**

As a first step, the UC retrieves a model "CAAScdfeaProductWithoutFEM" from DB and loads it and returns object of the Editor.

```
...
Dim myEditor As Editor
OpenProduct MyEditor
...
```

The function `OpenProduct` returns `MyEditor`, an Editor object. After searching and opening of "CAAScdfeaProductWithoutFEM" product from underlying database the current active editor is returned in `MyEditor`.

2. **Retrieves the root product**

As a next step, the UC retrieves the root product object from the returned editor.

```
...
Dim MyRootOcc As VPMRootOccurrence
Set MyRootOcc = MyEditor.ActiveObject

Dim MyRootProduct As VPMReference
Set MyRootProduct = MyRootOcc.PLMEEntity
...
```

3. **Retrieves the PartBody**

In this step UC retrieves the main body of the 3D shape aggregated by the root product.

```
...
Dim MyContext As PLMProductService
Set MyContext = MyEditor.GetService("PLMProductService")

Dim MyPart As Part
```

```

Dim MyPartInstance As VPMRepInstance
Dim MyBody As Body
For Each MyEntity In MyContext.EditedContent
    Dim MyRef As VPMReference
    Set MyRef = MyEntity

    Dim MyReps As VPMRepInstances
    Set MyReps = MyRef.RepInstances

    For Each MyRep In MyReps
        Dim MyRepRef As VPMRepReference
        Set MyRepRef = MyRep.ReferenceInstanceOf
        Dim attr As String
        attr = MyRepRef.GetAttributeValue("V_discipline")

        If (attr = "Design") Then
            Set MyPart = MyRepRef.GetItem("Part")
            Set MyPartInstance = MyRep
            Set MyBody = MyPart.MainBody
        End If
    Next
    ...

```

4. Creates the FEM representation and retrieve its root object

In this step UC creates a new FEM representation object. To perform the creation, a simulation representation factory have to be retrieved on the reference object on which the FEM rep will be aggregated.

```

...
Dim MyPrdRepFactory As SimPrdRepFactory
Set MyPrdRepFactory = MyRootProduct.GetItem("SimPrdRepFactory")
Dim MyFemRepRef As VPMRepReference
Set MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")
Dim MyFemRepRoot As SimFemRoot
Set MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")

```

The 3D Shape is linked to the FEM representation as assiocated shape.

```

Dim MyProductService As PLMProductService
Set MyProductService = MyEditor.GetService("PLMProductService")

Dim MyReference As Reference
Set MyReference = MyPart.CreateReferenceFromObject(MyBody)
Dim MyLink As AnyObject
Set MyLink = MyProductService.ComposeLink(MyRootOcc, MyPartInstance, MyReference)
MyFemRepRoot.AddAssociatedRep MyLink
...

```

5. Creates/Retrieves the group set

In this step UC retrieves, or creates if doesn't exist, the group set.

```

...
Dim MyGroupSet As SimGroups
Set MyGroupSet = MyFemRepRoot.GetSet("SimGroups")
...

```

6. Creates a geometric group

In this step UC creates a geometric group.

```

...
Dim MyGroupByAssociativity As SimGroup
Set MyGroupByAssociativity = MyGroupSet.Add("CATFmtGroupByAssociativity")
' Set the group support
Set MyPartSupp = MyPart.FindObjectByName("Fill.1")
Set MyReference = MyPart.CreateReferenceFromObject(MyPartSupp)
Set MyLink = MyProductService.ComposeLink(MyRootOcc, MyPartInstance, MyReference)
Dim MyLinkAccess As SimLinkAccess
Set MyLinkAccess = MyGroupByAssociativity.GetItem("SimLinkAccess")
MyLinkAccess.AddLink "MainSupport", MyLink
' Set the group type
MyGroupByAssociativity.SetGroupType ("CATFmtGroupTypeFace")
' Set the group name
MyGroupByAssociativity.Name = "Geometric Group.1"
...

```

7. Creates a proximity group

In this step UC creates a proximity group.

```

...
Dim MyGroupByProximity As SimGroup
Set MyGroupByProximity = MyGroupSet.Add("CATFmtGroupByProximity")
' Set the group support
Set MyPartSupp = MyPart.FindObjectByName("Fill.1")
Set MyReference = MyPart.CreateReferenceFromObject(MyPartSupp)
Set MyLink = MyProductService.ComposeLink(MyRootOcc, MyPartInstance, MyReference)
Set MyLinkAccess = MyGroupByAssociativity.GetItem("SimLinkAccess")
MyLinkAccess.AddLink "MainSupport", MyLink
' Set the group type
MyGroupByProximity.SetGroupType ("CATFmtGroupTypeFace")
' Set the group tolerance
MyGroupByProximity.SetAttributeValue "Tolerance", 3
' Set the group name
MyGroupByProximity.Name = "Proximity Group.1"
...

```

8. Remove the proximity group

In this step UC removes the proximity group.

```
...
MyGroupSet.Remove (2)
...
```

Managing FEM group features

This use case primarily focuses on the methodology to create/delete FEM groups.

Before you begin: Note that:

- You should first launch CATIA and import the CAAScdfeaProductWithoutFEM.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaModeling\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- You should open the product called CAASCDfeaProductWithoutFEM.
- Launch the structural scenario App and create a structural simulation.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaGroupsWithPythonSource.htm](#)

This use case can be divided in eight steps:

1. [Retrieves the simulation and its product](#)
2. [Retrieves the PartBody](#)
3. [Creates the FEM representation, Retrieves its root object and Associates the PartBody to this](#)
4. [Creates/Retrieves the group set](#)
5. [Creates a geometric group](#)
6. [Creates a proximity group](#)
7. [Remove the proximity group](#)

1. Retrieves the simulation and its product

As a first step, the UC retrieves the simulation and its product from the simulation editor

```
...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...

```

2. Retrieves the PartBody

In this step UC retrieves the main body of the 3D shape aggregated by the simulation's Model.

```
...
myModelRepInstances = myModel.RepInstances

for myPartRepInstance in myModelRepInstances:
    myPartRepInstanceRef = myPartRepInstance.ReferenceInstanceOf
    attr = myPartRepInstanceRef.GetAttributeValue("V_discipline")
    if attr == "Design":
        myPart = myPartRepInstanceRef.GetItem("Part")
        myPartInstance = myPartRepInstance
        myBody = myPart.MainBody
    ...

```

3. Creates the FEM representation, Retrieves its root object and Associates the PartBody to this

In this step UC creates a new FEM representation object.

To perform the creation, a simulation representation factory has to be retrieved on the reference object which will aggregate the FEM representation.

```
...
MyPrdRepFactory = myModel.GetItem("SimPrdRepFactory")
MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")
MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")
...

```

Now UC appends the main body previously retrieved to the associated shapes of the FEM representation.

First, the UC opens the model and retrieves the product editor and the root occurrence

```
...
oOpenService = CATIA.GetSessionService("PLMOpenService")
oOpenService.PLMOpen (myModel)
myProductEditor = CATIA.ActiveEditor
myRootOcc = myProductEditor.ActiveObject
...

```

Now it creates the link to associate with the FEM

```
...
MyReference = myPart.CreateReferenceFromObject(myBody)
myProdService = myProductEditor.GetService("PLMProductService")
MyLink = myProdService.ComposeLink(myRootOcc, myPartInstance, MyReference)
MyFemRepRoot.AddAssociatedRep (MyLink)
...

```

4. Creates/Retrieves the group set

In this step UC retrieves, or creates if doesn't exist, the group set.

```
...
MyGroupSet = MyFemRepRoot.GetSet("SimGroups")
...
```

5. Creates a geometric group

In this step UC creates a geometric group.

```
...
MyGroupByAssociativity = MyGroupSet.Add("CATFmtGroupByAssociativity")

# Set the group support
MyPartSupp = myPart.FindObjectByName("Fill.1")
MyReference = myPart.CreateReferenceFromObject(MyPartSupp)
MyLink = myProductService.ComposeLink(MyRootOcc, MyPartInstance, MyReference)

MyLinkAccess = MyGroupByAssociativity.GetItem("SimLinkAccess")
MyLinkAccess.AddLink ("MainSupport", MyLink)

# Set the group type
MyGroupByAssociativity.SetGroupType("CATFmtGroupTypeFace")

# Set the group name
MyGroupByAssociativity.Name = "Geometric Group.1"
...
```

6. Creates a proximity group

In this step UC creates a proximity group.

```
...
MyGroupByProximity = MyGroupSet.Add("CATFmtGroupByProximity")

# Set the group support
MyPartSupp = myPart.FindObjectByName("Fill.1")
MyReference = myPart.CreateReferenceFromObject(MyPartSupp)
MyLink = myProductService.ComposeLink(MyRootOcc, MyPartInstance, MyReference)
MyLinkAccess = MyGroupByAssociativity.GetItem("SimLinkAccess")
MyLinkAccess.AddLink ("MainSupport", MyLink)

# Set the group type
MyGroupByProximity.SetGroupType("CATFmtGroupTypeFace")

# Set the group tolerance
MyGroupByProximity.SetAttributeValue ("Tolerance", 3)

# Set the group name
MyGroupByProximity.Name = "Proximity Group.1"
...
```

7. Remove the proximity group

In this step UC removes the proximity group.

```
...
MyGroupSet.Remove (2)
...
```

Creating a XRep and managing its datas

This use case primarily focuses on the methodology to create a XRep and to manage data files.

Before you begin: Note that:

- You should first launch CATIA and import the CAAScdfeaProductWithoutFEM.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaModeling\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- You should also copy into the folder pointed by the "TEMP" environment variable the "RootProduct.inp.txt" file supplied in the same folder as the 3dxml file. **The file extension must be changed to "inp" instead of "txt".**
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaCreateXRepSource.htm](#)

This use case can be divided in nine steps:

1. [Searches and opens model which is named as "CAAScdfeaProductWithoutFEM"](#)
2. [Retrieves the root product](#)
3. [Retrieves the PartBody](#)
4. [Creates a new XRep](#)
5. [Sets the data file name associated to the XRep](#)
6. [Imports the data file in the XRep](#)
7. [Generates the nav rep and import it into the XRep](#)
8. [Adds a relation in the XRep](#)
9. [Exports the data of the XRep](#)

1. Searches and opens model which is named as "CAAScdfeaProductWithoutFEM"

As a first step, the UC retrieves a model "CAAScdfeaProductWithoutFEM" from DB and loads it and returns object of the Editor.

```
...
Dim myEditor As Editor
OpenProduct MyEditor
...
```

The function `OpenProduct` returns `MyEditor`, an Editor object. After searching and opening of "CAAScdfeaProductWithoutFEM" product from underlying database the current active editor is returned in `MyEditor`.

2. Retrieves the root product

As a next step, the UC retrieves the root product object from the returned editor.

```
...
Dim MyRootOcc As VPMRootOccurrence
Set MyRootOcc = MyEditor.ActiveObject

Dim MyRootProduct As VPMReference
Set MyRootProduct = MyRootOcc.PLMEEntity
...
```

3. Retrieves the PartBody

In this step UC retrieves the main body of the 3D shape aggregated by the root product.

```
...
Dim MyContext As PLMProductService
Set MyContext = MyEditor.GetService("PLMProductService")

Dim MyPart As Part
Dim MyPartInstance As VPMRepInstance
Dim MyBody As Body
For Each MyEntity In MyContext.EditedContent
    Dim MyRef As VPMReference
    Set MyRef = MyEntity

    Dim MyReps As VPMRepInstances
    Set MyReps = MyRef.RepInstances

    For Each MyRep In MyReps
        Dim MyRepRef As VPMRepReference
        Set MyRepRef = MyRep.ReferenceInstanceOf
        Dim attr As String
        attr = MyRepRef.GetAttributeValue("V_discipline")

        If (attr = "Design") Then
            Set MyPart = MyRepRef.GetItem("Part")
            Set MyPartInstance = MyRep
            Set MyBody = MyPart.MainBody
        End If
    Next
...

```

4. Creates a new XRep

In this step UC creates a new XRep object. To perform the creation, a simulation representation factory have to be retrieved on the reference object on which the XRep will be aggregated.

```
...
Dim MyPrdRepFactory As SimPrdRepFactory
Set MyPrdRepFactory = MyRootProduct.GetItem("SimPrdRepFactory")

Dim MyXRepRef As VPMRepReference
Set MyXRepRef = MyPrdRepFactory.CreatePrdRep("XRep")

Dim MyXRep As SimXRep
Set MyXRep = MyXRepRef.GetItem("SimXRep")
...
```

5. Sets the data file name associated to the XRep

In this step UC defines the name of the data file.

```
...
MyXRep.FileName = "RootProduct.inp"
...
```

6. Imports the data file in the XRep

In this step UC imports datas from the data file.

```
...
Dim sFolderPath As String
sFolderPath = CATIA.SystemService.Environ("TEMP")
MyXRep.ImportFile sFolderPath, True
...
```

7. Generates the nav rep and import it into the XRep

In this step UC generates the cgr file and imports it into the XRep.

```
...
Dim sFilePath As String
sFilePath = CATIA.SystemService.Environ("TEMP") & "\RootProduct.inp"
MyXRep.GenerateNavRep sFilePath, "METER"
...
```

8. Adds a relation in the XRep

In this step UC add the associated shape of the new representation.

```
...
MyFemRepRoot.RemoveAssociatedRep MyLink
bHasAnAssociatedRep = MyFemRepRoot.HasAnAssociatedRep
If (bHasAnAssociatedRep = True) Then
    MsgBox "Error while removing the associated shape from the FEM Rep!"
    Exit Sub
End If
```

...

9. Exports the data of the XRep to a data file

In this step UC exports the XRep datas into an "inp" file.

```
...
MyXRep.FileName = "RootProductOut.inp"
MyXRep.ExportFile sFolderPath
...
```

Creating a XRep and managing its datas

This use case primarily focuses on the methodology to create a XRep and to manage data files.

Before you begin: Note that:

- You should first launch CATIA and import the CAAScdfeaProductWithoutFEM.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaModeling\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- You should also copy into the folder pointed by the "TEMP" environment variable the "RootProduct.inp.txt" file supplied in the same folder as the 3dxml file. **The file extension must be changed to "inp" instead of "txt".**
- You should open the product called CAASCDfeaProductWithoutFEM.
- Launch the structural scenario App and create a structural simulation.

Where to find the macro: [CAAScdfeaCreateXRepWithPythonSource.htm](#)

This use case can be divided in eight steps:

1. [Retrieves the simulation and its product](#)
2. [Retrieves the PartBody](#)
3. [Creates a new XRep](#)
4. [Sets the data file name associated to the XRep](#)
5. [Imports the data file in the XRep](#)
6. [Adds a relation in the XRep](#)
7. [Exports the data of the XRep](#)

1. Retrieves the simulation and its product

As a first step, the UC retrieves the simulation and its product from the simulation editor.

```
...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...
```

Then opens the simulation model and retrieves the root occurrence and the root product.

```
...
oOpenService = CATIA.GetSessionService("PLMOpenService")
oOpenService.PLMOpen (myModel)

myProductEditor = CATIA.ActiveEditor
MyProductService = myProductEditor.GetService("PLMProductService")

myRootOcc = MyProductService.RootOccurrence
MyRootProduct = myRootOcc.PLMEntity
...
```

2. Retrieves the PartBody

In this step UC retrieves the main body of the 3D shape aggregated by the simulation's Model.

```
...
MyEntities = MyProductService.EditedContent

for MyEntity in MyEntities:
    MyRef = MyEntity
    MyReps = MyRef.RepInstances
    for MyRep in MyReps :
        MyRepRef = MyRep.ReferenceInstanceOf
        attr = MyRepRef.GetAttributeValue("V_discipline")
        if attr == "Design":
            myPart = MyRepRef.GetItem("Part")
            myPartInstance = MyRep
            myBody = myPart.MainBody
...

```

3. Creates a new XRep

In this step UC creates a new XRep object. To perform the creation, a simulation representation factory have to be retrieved on the reference object on which the XRep will be aggregated.

```
...
MyPrdRepFactory = MyRootProduct.GetItem("SimPrdRepFactory")
MyXRepRef = MyPrdRepFactory.CreatePrdRep("XRep")
MyXRep = MyXRepRef.GetItem("SimXRep")
...
```

4. Sets the data file name associated to the XRep

In this step UC defines the name of the data file.

```
...
MyXRep.FileName = "RootProduct.inp"
...
```

5. Imports the data file in the XRep

In this step UC imports datas from the data file.

```
...
sFolderPath = CATIA.SystemService.Environ("TEMP")
MyXRep.ImportFile (sFolderPath, True)
...
```

6. Adds a relation in the XRep

In this step UC add the associated shape of the new representation.

```
...
MyLink = MyProductService.ComposeLink(myRootOcc, myPartInstance, None)
MyXRep.AddRelation(MyLink, win32com.client.constants.SimXRepTo3DShapeRelation)
...
```

7. Exports the data of the XRep

In this step UC exports the XRep data into an "inp" file.

```
...
MyXRep.FileName = "RootProductOut.inp"
MyXRep.ExportFile(sFolderPath)
...
```

Creation Late Types for Model Features

Technical Article

Abstract

The following reference article documents the different late types that can be used to create model features.

Related Topics

- [SimProperties Collection](#)
- [SimMCXProperties Collection](#)
- [SimAbstractions Collection](#)
- [Multiphysics Model Creation Object Model Map](#)

Mesh Parts Late Types

For further information about the mesh parts late types refer to the article "[Mesh Parts and Their Attributes](#)"

Properties Late Types

Properties creation is managed by the **SimProperties** collection by calling the `Add` method from it.

A "late type" must be given to indicate which type of property is to be created:

Section	Late type
1D Link Section	Sim1DLinkSection
Beam Section	SimBeamSection
Composite Shell Section	SimCompositeShellSection
Continuum Shell Section	SimContinuumShellSection
Fluid Section	SimFluidSection
Gasket Property	SimGasketProperty
Meshed Bolt	SimMeshedBolt
Shell Section	SimShellSection
Solid Section	SimSolidSection
Cohesive Property	SimCohesiveProperty

Connection Properties Late Types

Connection properties creation is managed by the **SimMCXProperties** collection by calling the `Add` method from it.

A "late type" must be given to indicate which type of connection property is to be created:

Connection property	Late type
Connector	SimConnector
Coupling	SimCoupling
Line Fastener	SimLineFastener♦
Point Fastener	SimPointFastener♦
Rigid Connection	SimRigidConnection
Spring	SimSpring
Tie	SimTie♦
Virtual Bolt	SimVirtualBolt
Virtual Pin	SimVirtualPin

♦ Available with the *RBM - Simulation Scripting* product only.

Behaviors Late Types

Behaviors creation is managed by the **SimBehaviors** collection by calling the `Add` method from it.

A "late type" must be given to indicate which type of behavior is to be created:

Behavior	Late type
Beam Profile	SimBeamProfile

Abstractions Late Types

Abstractions creation is managed by the **SimAbstractions** collection by calling the `Add` method from it.

A "late type" must be given to indicate which type of abstraction is to be created:

Abstraction	Late type
Acoustic Absorber	SimAcousticAbsorber
Acoustic Coupling	SimAcousticCoupling
Analytic Rigid Surface	SimAnalyticRigidSurface
Mass Rotary Inertia	SimMassRotaryInertia
Cyclic Symmetry	SimCyclicSymmetry
Fluid Cavity	SimFluidCavity
Mass Per Length	SimMassPerLength
Mass Per Area	SimMassPerArea
Mass Per Volume	SimMassPerVolume
Rigid Body Constraint	SimRigidBodyConstraint
Surface Fluid Cavity	SimSurfaceFluidCavity

History

Version: 1 [Jan 2015] Document created

Managing FEM Section Features

This use case primarily focuses on the methodology to create/delete FEM sections.

Before you begin:

- Launch CATIA and then import and open the `CAAScdfeaPumpModel.3dxml` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaSectionsSource.htm](#)

This use case can be divided in eight steps:

1. [Retrieving the active product](#)
2. [Retrieving the parts](#)
3. [Retrieving the publications](#)
4. [Creating the FEM representation and retrieving its root object](#)
5. [Retrieving the properties set](#)
6. [Creating individual properties](#)
7. [Setting individual attributes](#)
8. [Retrieving/Remove individual properties](#)
9. [Retrieving Shell Section attributes : Thickness, Integration Method...](#)

1. Retrieving the active product

As a first step, the UC retrieves the active product.

```
...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor

Dim myProdService As PLMProductService
Set myProdService = myEditor.GetService("PLMProductService")

Dim myEntities As PLMEntities
Set myEntities = myProdService.EditedContent

Dim myRootProduct As VPMReference
Set myRootProduct = myEntities.Item(1)
...
```

2. Retrieving the Parts

In this step UC retrieves the 3D shapes aggregated by the root product.

```
...
Dim myProductInstance As VPMInstance
Dim myPartInstance as VPMRepInstance
Dim myPart as Part
For Each myProductInstance in myRootProduct.Instances
    Dim myProductInstanceRef as VPMReference
    Set myProductInstanceRef = myProductInstance.ReferenceInstanceOf
    For Each myPartInstance in myProductInstanceRef.RepInstances
        Dim myPartInstanceRef as VPMRepReference
    
```

```

    Set myPartInstanceRef = myPartInstance.ReferenceInstanceOf
    If (myPartInstanceRef.GetAttributeValue("V_discipline") = "Design") Then
        Set myPart = myPartInstanceRef.GetItem("Part")
    End If
    Next
Next
...

```

3. Retrieving the publications

In this step UC retrieves the publications aggregated by the root product.

```

...
Dim myPublications As VPMPublications
Set myPublications = myRootProduct.Publications

Dim myPublication as VPMPublication
Set myPublication = myPublications.GetItem("Pipe")
...

```

4. Creating the FEM representation and retrieving its root object

In this step UC creates a new FEM representation object. To perform the creation, a simulation representation factory have to be retrieved on the reference object on which the FEM rep will be aggregated.

```

...
Dim myPrdRepFactory As SimPrdRepFactory
Set myPrdRepFactory = myRootProduct.GetItem("SimPrdRepFactory")

Dim myFemRepRef As VPMRepReference
Set myFemRepRef = myPrdRepFactory.CreatePrdRep("FEM")

Dim myFemRepRoot As SimFemRoot
Set myFemRepRoot = myFemRepRef.GetItem("SimFemRoot")

```

The 3D Shapes are linked to the FEM representation as associated shapes.

```

Dim myRootOccurrence as VPMRootOccurrence
Set myRootOccurrence = myProdService.RootOccurrence

Dim myRepInstance As VPMRepInstance
Set myRepInstance = Nothing ' Because we use publications

Dim myLink As AnyObject
Set myLink = myProdService.ComposeLink(myRootOccurrence, myRepInstance, myPublication)
MyFemRepRoot.AddAssociatedRep myLink
...

```

5. Retrieving the properties set

In this step UC retrieves the properties set.

```

...
Dim myPropertiesSet As SimProperties
Set myPropertiesSet = myFemRepRoot.GetSet("SimProperties")
...

```

6. Creating individual properties

In this step UC creates a solid section and a shell section feature.

```

...
Dim mySolidSection As SimSolidSection
Set mySolidSection = myPropertiesSet.Add("SimSolidSection")

Dim myShellSection As SimShellSection
Set myShellSection = myPropertiesSet.Add("SimShellSection")
...

```

You can find the available late types for properties creation in the [Creation Late Types for Model Features](#) article.

7. Setting individual attributes

In this step UC sets individual attributes of the sections.

```

...
' Sets the section support
Dim myLinkAccess As SimLinkAccess
Set myLinkAccess = myShellSection.GetItem("SimLinkAccess")
myLinkAccess.AddLink "MainSupport", myLink

' Sets the material behavior
myShellSection.MaterialBehaviorByName = "Behavior.1"

' Sets the orientation
Dim myOrientation As SimOrientation
Set myOrientation = myShellSection.Orientation

Dim myAxisSystem As SimAxisSystem
Set myAxisSystem = myOrientation.AxisSystem
myAxisSystem.CoordinateType = SimAxisSystemCartesian

myOrientation.OrientationFlag = True
myOrientation.AxisOfRotation = SimOrientationAxis2
myOrientation.AngleOfRotation = 90

' Sets others attributes
myShellSection.IntegrationScheme = SimShellSectionSimpsonIntegration

```

```

myShellSection.PoissonMethod = SimShellSectionPoissonDefault
myShellSection.UniformThickness = 0.002
...

```

8. Retrieving/Remove individual properties

In this step UC retrieves the created properties and remove the solid section.

```

...
Dim myRetrievedSolidSection As SimSolidSection
Set myRetrievedSolidSection = myPropertiesSet.Item(1)

Dim myRetrievedShellSection As SimShellSection
Set myRetrievedShellSection = myPropertiesSet.Item(2)

myPropertiesSet.Remove(1)
...

```

9. Retrieving Shell Section attributes : Thickness, Integration Method...

In this step UC retrieves the properties attributes.

```

...
Dim myIntegrationScheme As SimShellSectionIntegrationScheme
myIntegrationScheme = myRetrievedShellSection.IntegrationScheme

Dim myNumberOfIntegrationPoints As Long
myNumberOfIntegrationPoints = myRetrievedShellSection.NumIntPoints

If myRetrievedShellSection.UniformThicknessFlag Then
    Dim myUniformThickness As Double
    myUniformThickness = myRetrievedShellSection.UniformThickness
End If

Dim myMaterialBehavior As MaterialBehavior
Set myMaterialBehavior = myRetrievedShellSection.MaterialBehavior

Set myOrientation = myRetrievedShellSection.Orientation

If myOrientation.OrientationFlag Then
    Dim myRotationAxis As SimOrientationAxisOfRotation
    myRotationAxis = myOrientation.AxisOfRotation

    Dim myAngleOfRotation As Double
    myAngleOfRotation = myOrientation.AngleOfRotation
End If
...

```

Managing FEM Section Features

This use case primarily focuses on the methodology to create/delete FEM sections.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Launch the structural scenario App and create a structural simulation.

Where to find the macro: [CAAScdfeaSectionsWithPythonSource.htm](#)

This use case can be divided in eight steps:

1. [Retrieves the simulation and its product](#)
2. [Retrieves the Root occurrence and the root product](#)
3. [Retrieves the publications](#)
4. [Creates the FEM representation and retrieving its root object](#)
5. [Retrieves the properties set](#)
6. [Creates individual properties](#)
7. [Sets individual attributes](#)
8. [Retrieves/Removes individual properties](#)
9. [Retrieves Shell Section attributes : Thickness, Integration Method...](#)

1. Retrieves the simulation and its product

As a first step, the UC retrieves the simulation and its product from the simulation editor.

```

...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...

```

2. Retrieves the Root occurrence and the root product

In this step UC opens the simulation model and retrieves the root occurrence and the root product.

```

...
oOpenService = CATIA.GetSessionService("PLMOpenService")
oOpenService.PLMOpen (myModel)

myProductEditor = CATIA.ActiveEditor
myProductService = myProductEditor.GetService("PLMProductService")

myProductEntities = myProductService.EditedContent
myRootProduct = myProductEntities.Item(1)

```

```
myRootOccurrence = myProductService.RootOccurrence
...
```

3. Retrieves the publications

In this step UC retrieves the publications aggregated by the root product.

```
...
myPublications = myRootProduct.Publications
myPublication = myPublications.GetItem("Pipe")
...
```

4. Creates the FEM representation and retrieving its root object

In this step UC creates a new FEM representation object. To perform the creation, a simulation representation factory have to be retrieved on the reference object on which the FEM rep will be aggregated.

```
...
MyPrdRepFactory = myRootProduct.GetItem("SimPrdRepFactory")
MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")
MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")
...
```

The 3D Shapes are linked to the FEM representation as associated shapes.

```
...
myLink = myProductService.ComposeLink(myRootOccurrence, None, myPublication)
MyFemRepRoot.AddAssociatedRep (myLink)
...
```

5. Retrieves the properties set

In this step UC retrieves the properties set.

```
...
myPropertiesSet = MyFemRepRoot.GetSet("SimProperties")
...
```

6. Creates individual properties

In this step UC creates a solid section and a shell section feature.

```
...
mySolidSection = myPropertiesSet.Add("SimSolidSection")
myShellSection = myPropertiesSet.Add("SimShellSection")
...
```

You can find the available late types for properties creation in the [Creation Late Types for Model Features](#) article.

7. Sets individual attributes

In this step UC sets individual attributes of the sections.

```
...
# Sets the section support
myLinkAccess = myShellSection.GetItem("SimLinkAccess")
myLinkAccess.AddLink ("MainSupport", myLink)

# Sets the material behavior
myShellSection.MaterialBehaviorByName = "Behavior.1"

# Sets the orientation
myOrientation = myShellSection.Orientation

myAxisSystem = myOrientation.AxisSystem
myAxisSystem.CoordinateType = win32com.client.constants.SimAxisSystemCartesian

myOrientation.OrientationFlag = True
myOrientation.AxisOfRotation = win32com.client.constants.SimOrientationAxis2
myOrientation.AngleOfRotation = 90

# Sets others attributes
myShellSection.IntegrationScheme = win32com.client.constants.SimShellSectionSimpsonIntegration
myShellSection.PoissonMethod = win32com.client.constants.SimShellSectionPoissonDefault
myShellSection.UniformThickness = 0.002
...
```

8. Retrieves/Removes individual properties

In this step UC retrieves the created properties and remove the solid section.

```
...
myRetrievedSolidSection = myPropertiesSet.Item(1)
myRetrievedShellSection = myPropertiesSet.Item(2)
myPropertiesSet.Remove(1)
...
```

9. Retrieves Shell Section attributes : Thickness, Integration Method...

In this step UC retrieves the properties attributes.

```
...
myIntegrationScheme = myRetrievedShellSection.IntegrationScheme
myNumberOfIntegrationPoints = myRetrievedShellSection.NumIntPoints
```

```

bHasUniformThickness = myRetrievedShellSection.UniformThicknessFlag
if bHasUniformThickness == True:
    myUniformThickness = myRetrievedShellSection.UniformThickness

myMaterialBehavior = myRetrievedShellSection.MaterialBehavior

myOrientation = myRetrievedShellSection.Orientation

bHasOrientation = myOrientation.OrientationFlag
if bHasOrientation == True:
    myRotationAxis = myOrientation.AxisOfRotation
    myAngleOfRotation = myOrientation.AngleOfRotation
...

```

Creating Numbering Specifications

This use case primarily focuses on the methodology to create nodes and elements numbering specifications for an assembly.

Before you begin: Note that:

- You should first launch CATIA and import the CAAScdfeaAssemblyForNumbering.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaModeling\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: [CAAScdfeaNumberingSource.htm](#)

This use case can be divided in eleven steps:

1. [Retrieves the active editor](#)
2. [Retrieves the root product](#)
3. [Retrieves the assembly FEM Rep](#)
4. [Creates global numbering specifications](#)
5. [Creates a link between the assembly FEM and the first occurrence FEM](#)
6. [Sets numbering offsets](#)
7. [Retrieves the mesh part of the first occurrence FEM](#)
8. [Creates mesh part numbering specifications](#)
9. [Retrieves the group of the first occurrence FEM](#)
10. [Creates group numbering specifications](#)
11. [Updates the assembly FEM Rep](#)

1. Retrieves the active editor

As a first step, the UC retrieves the active editor.

```

...
Dim MyEditor As Editor
Set MyEditor = CATIA.ActiveEditor
...

```

2. Retrieves the root product

In this step, the UC retrieves the root product from the active editor.

```

...
Dim MyRootOcc As VPMRootOccurrence
Set MyRootOcc = MyEditor.ActiveObject

Dim MyRootProduct As VPMReference
Set MyRootProduct = MyRootOcc.PLMEentity
...

```

3. Retrieves the assembly FEM Rep

In this step, the UC retrieves the FEM Rep of the assembly.

```

...
Dim MyProductService As PLMProductService
Set MyProductService = MyEditor.GetService("PLMProductService")

Dim MyFEMRoot As SimFemRoot
For Each MyEntity In MyProductService.EditedContent
    Dim MyRef As VPMReference
    Set MyRef = MyEntity

    Dim MyReps As VPMRepInstances
    Set MyReps = MyRef.RepInstances

    For Each MyRep In MyReps
        Dim MyRepRef As VPMRepReference
        Set MyRepRef = MyRep.ReferenceInstanceOf
        Dim attr As String
        attr = MyRepRef.GetAttributeValue("V_discipline")

        If (attr = "FEM") Then
            Set MyFEMRoot = MyRepRef.GetItem("SimFemRoot")
        End If
    Next
Next
...

```

4. Creates global numbering specifications

In this step, the UC creates global nodes and elements numbering specifications for the assembly.

```

...
Dim MyNumberingEntity As SimNumbering
Set MyNumberingEntity = MyFEMRoot.GetItem("SimNumbering")
MyNumberingEntity.SetAttributeValue "NodesLowerBoundValue", 1
MyNumberingEntity.SetAttributeValue "NodesUpperBoundValue", 1000000

```

```

MyNumberingEntity.SetAttributeValue "ElementsLowerBoundValue", 1
MyNumberingEntity.SetAttributeValue "ElementsUpperBoundValue", 2000000
...

```

5. Creates a link between the assembly FEM and the first occurrence FEM

In this step, the UC retrieves the FEM Rep which is aggregated by the first product occurrence. Then the UC creates a path element that will be used to create the link between the retrieved FEM Rep and the assembly FEM Rep.

```

...
Dim MyRootOccurrences As VPMOccurrences
Set MyRootOccurrences = MyRootOcc.Occurrences
Dim MyOccurrence As VPMOccurrence
Set MyOccurrence = MyRootOccurrences.Item(1)

Dim MyRepInstances As VPMRepInstances
Set MyRepInstances = MyOccurrence.InstanceOccurrenceOf.ReferenceInstanceOf.RepInstances

Dim MyFirstFEMRoot As SimFemRoot
Dim MyFirstLink As AnyObject
For Each MyRep In MyRepInstances
    Set MyRepRef = MyRep.ReferenceInstanceOf
    attr = MyRepRef.GetAttributeValue("V_discipline")
    If (attr = "FEM") Then
        Set MyFirstFEMRoot = MyRepRef.GetItem("SimFemRoot")
        Set MyFirstLink = MyProductService.ComposeLink(MyOccurrence, MyRep, Nothing)
    End If
Next
...

```

6. Sets numbering offsets

In this step, the UC sets nodes and elements numbering offsets of the first occurrence FEM

```

...
MyFEMRoot.SetNumberingOffsetValue "nodes", MyFirstLink, 500000
MyFEMRoot.SetNumberingOffsetValue "elements", MyFirstLink, 600000
...

```

7. Retrieves the mesh part of the first occurrence FEM

In this step, the UC retrieves the mesh part of the first occurrence FEM.

```

...
Dim MyMeshSet As SimMeshSet
Set MyMeshSet = MyFirstFEMRoot.GetSet("SimNodesElements")
Dim MyMeshParts As SimMeshParts
Set MyMeshParts = MyMeshSet.MeshParts
Dim MyMeshPart As SimMeshPart
Set MyMeshPart = MyMeshParts.Item(1)
...

```

8. Creates mesh part numbering specifications

In this step, the UC creates nodes and elements numbering specifications for the mesh part.

```

...
Set MyNumberingEntity = MyMeshPart.GetItem("SimNumbering")
MyNumberingEntity.SetAttributeValue "NodesLowerBoundValue", 100001
MyNumberingEntity.SetAttributeValue "NodesUpperBoundValue", 102000
MyNumberingEntity.SetAttributeValue "ElementsLowerBoundValue", 103001
MyNumberingEntity.SetAttributeValue "ElementsUpperBoundValue", 105000
...

```

9. Retrieves the group of the first occurrence FEM

In this step, the UC retrieves the group of the first occurrence FEM.

```

...
Dim MyGroups As SimGroups
Set MyGroups = MyFirstFEMRoot.GetSet("SimGroups")
Dim MyGroup As SimGroup
Set MyGroup = MyGroups.Item(1)
...

```

10. Creates group numbering specifications

In this step, the UC creates nodes numbering specifications for the group.

```

...
Set MyNumberingEntity = MyGroup.GetItem("SimNumbering")
MyNumberingEntity.SetAttributeValue "NodesLowerBoundValue", 105101
MyNumberingEntity.SetAttributeValue "NodesUpperBoundValue", 105200
...

```

11. Updates the assembly FEM Rep

In this step, the UC updates the assembly FEM Rep.

```

...
MyFEMRoot.Update
...

```

Creating Numbering Specifications

This use case primarily focuses on the methodology to create nodes and elements numbering specifications for an assembly.

Before you begin: Note that:

- You should first launch CATIA and import the CAAScdfeaAssemblyForNumbering.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaModeling\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- You should open the product called CAAScdfeaAssemblyForNumbering.
- Launch the structural scenario App and create a structural simulation.

Where to find the macro: [CAAScdfeaNumberingWithPythonSource.htm](#)

This use case can be divided in eleven steps:

1. [Retrieve the simulation and its product](#)
2. [Retrieves the assembly FEM Rep](#)
3. [Creates global numbering specifications](#)
4. [Creates a link between the assembly FEM and the first occurrence FEM](#)
5. [Sets numbering offsets](#)
6. [Retrieves the mesh part of the first occurrence FEM](#)
7. [Creates mesh part numbering specifications](#)
8. [Retrieves the group of the first occurrence FEM](#)
9. [Creates group numbering specifications](#)
10. [Updates the assembly FEM Rep](#)

1. Retrieve the simulation and its product

As a first step, the UC retrieves the active editor and retrieves the simulation model.

```
...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...
```

Then opens the simulation model and retrieves the root occurrence.

```
oOpenService = CATIA.GetSessionService("PLMOpenService")
oOpenService.PLMOpen (myModel)

myProductEditor = CATIA.ActiveEditor
MyProductService = myProductEditor.GetService("PLMProductService")

myRootOcc = MyProductService.RootOccurrence
...
```

2. Retrieves the assembly FEM Rep

In this step, the UC retrieves the FEM Rep of the assembly.

```
...
myProductEntities = MyProductService.EditedContent

for myProductEntity in myProductEntities:
    myRef = myProductEntity
    myReps = myRef.RepInstances

    for myRep in myReps:
        myRepRef = myRep.ReferenceInstanceOf
        attr = myRepRef.GetAttributeValue("V_discipline")
        if attr == "FEM":
            myFEMRoot = myRepRef.GetItem("SimFemRoot")
...

```

3. Creates global numbering specifications

In this step, the UC creates global nodes and elements numbering specifications for the assembly.

```
...
MyNumberingEntity = myFEMRoot.GetItem("SimNumbering")
MyNumberingEntity.SetAttributeValue ("NodesLowerBoundValue", 1)
MyNumberingEntity.SetAttributeValue ("NodesUpperBoundValue", 1000000)
MyNumberingEntity.SetAttributeValue ("ElementsLowerBoundValue", 1)
MyNumberingEntity.SetAttributeValue ("ElementsUpperBoundValue", 2000000)
...
```

4. Creates a link between the assembly FEM and the first occurrence FEM

In this step, the UC retrieves the FEM Rep which is aggregated by the first product occurrence. Then the UC creates a path element that will be used to create the link between the retrieved FEM Rep and the assembly FEM Rep.

```
...
MyRootOccurrences = myRootOccOccurrences
MyOccurrence = MyRootOccurrences.Item(1)

MyRepInstances = MyOccurrence.InstanceOccurrenceOf.ReferenceInstanceOf.RepInstances

for MyRep2 in MyRepInstances:
    MyRepRef2 = MyRep2.ReferenceInstanceOf
    attr = MyRepRef2.GetAttributeValue("V_discipline")
    if attr == "FEM":
        MyFirstFEMRoot = MyRepRef2.GetItem("SimFemRoot")
        MyFirstLink = MyProductService.ComposeLink(MyOccurrence, MyRep2, None)
...

```

5. Sets numbering offsets

In this step, the UC sets nodes and elements numbering offsets of the first occurrence FEM

```
...
myFEMRoot.SetNumberingOffsetValue ("nodes", MyFirstLink, 500000)
myFEMRoot.SetNumberingOffsetValue ("elements", MyFirstLink, 600000)
...
```

6. Retrieves the mesh part of the first occurrence FEM

In this step, the UC retrieves the mesh part of the first occurrence FEM.

```
...
MyMeshSet = MyFirstFEMRoot.GetSet("SimNodesElements")
MyMeshParts = MyMeshSet.MeshParts
MyMeshPart = MyMeshParts.Item(1)
...
```

7. Creates mesh part numbering specifications

In this step, the UC creates nodes and elements numbering specifications for the mesh part.

```
...
MyNumberingEntity = MyMeshPart.GetItem("SimNumbering")
MyNumberingEntity.SetValue ("NodesLowerBoundValue", 100001)
MyNumberingEntity.SetValue ("NodesUpperBoundValue", 102000)
MyNumberingEntity.SetValue ("ElementsLowerBoundValue", 103001)
MyNumberingEntity.SetValue ("ElementsUpperBoundValue", 105000)
...

```

8. Retrieves the group of the first occurrence FEM

In this step, the UC retrieves the group of the first occurrence FEM.

```
...
MyGroups = MyFirstFEMRoot.GetSet("SimGroups")
MyGroup = MyGroups.Item(1)
...
```

9. Creates group numbering specifications

In this step, the UC creates nodes numbering specifications for the group.

```
...
MyNumberingEntity = MyGroup.GetItem("SimNumbering")
MyNumberingEntity.SetValue ("NodesLowerBoundValue", 105101)
MyNumberingEntity.SetValue ("NodesUpperBoundValue", 105200)
...

```

10. Updates the assembly FEM Rep

In this step, the UC updates the assembly FEM Rep.

```
...
myFEMRoot.Update
...
```

Creating a Connection Property

This use case primarily focuses on the methodology to create a connection property.

Before you begin:

- Launch CATIA and then import and open the CAASeqFeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAASeqFeaScenario\samples\`, where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAASeqFeaConnectionsSource.htm](#)

This use case can be divided into seven steps:

1. [Retrieving the active product](#)
2. [Retrieving the publications and composing links](#)
3. [Creating a rigid engineering connection](#)
4. [Creating the FEM representation and retrieving its root object](#)
5. [Retrieving the MCX property manager](#)
6. [Creating a spring connection property](#)
7. [Setting the spring connection attributes](#)

1. Retrieving the active product

As a first step, the UC retrieves the active product.

```
...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor

Dim myProdService As PLMProductService
Set myProdService = myEditor.GetService("PLMProductService")

Dim myEntities As PLMEntities
Set myEntities = myProdService.EditedContent

Dim myRootProduct As VPMReference
Set myRootProduct = myEntities.Item(1)
...

```

2. Retrieving the publications and composing links

In this step UC retrieves the publications aggregated by the root product and composes links.

```
...
Dim myPublications As VPMPublications
Set myPublications = myRootProduct.Publications

Dim myCoverContactFace As VPMPublication
Set myCoverContactFace = myPublications.GetItem("Cover_Contact_Face")

' Composing link to the publication

Dim myRootOccurrence As VPMRootOccurrence
Set myRootOccurrence = myProdService.RootOccurrence

Dim myRepInstance As VPMRepInstance
Set myRepInstance = Nothing ' Because we use publications

Dim myLinkToCoverContactFace As AnyObject
Set myLinkToCoverContactFace = myProdService.ComposeLink(myRootOccurrence, myRepInstance, myCoverContactFace)
...
```

3. Creating a rigid engineering connection

In this step UC creates a rigid engineering connection.

For further information about creating a rigid engineering connection refer to the article [Creating an Assembly Constraint Between Two Instances of a Basic Pad](#).

4. Creating the FEM representation and retrieving its root object

In this step UC creates a new FEM representation object. To perform the creation, a simulation representation factory has to be retrieved on the reference object on which the FEM rep will be aggregated.

```
...
Dim myPrdRepFactory As SimPrdRepFactory
Set myPrdRepFactory = myRootProduct.GetItem("SimPrdRepFactory")

Dim myFemRepRef As VPMRepReference
Set myFemRepRef = myPrdRepFactory.CreatePrdRep("FEM")

Dim myFemRepRoot As SimFemRoot
Set myFemRepRoot = myFemRepRef.GetItem("SimFemRoot")
...
```

5. Retrieving the MCX property manager

In this step UC retrieves the MCX property manager.

```
...
Dim myMCXProperties As SimMCXProperties
Set myMCXProperties = myNewRigidEngineeringConnection.GetItem("SimMCXProperties")
...
```

6. Creating a spring connection property

In this step UC creates a spring connection property.

```
' Creation of the spring connection property

Dim mySpringConnection As SimSpring
Set mySpringConnection = myMCXProperties.Add("SimSpring", myFemRepRoot)

' Setting of the Spring connection property supports

Dim myLinkAccess As SimLinkAccess
Set myLinkAccess = mySpringConnection.GetItem("SimLinkAccess")

myLinkAccess.AddLink "PRD1", myLinkToCoverContactFace ' PRD1 attribute is for main surface
myLinkAccess.AddLink "PRD2", myLinkToPumpHousingContactFace ' PRD2 attribute is for secondary surface
...
```

You can find the available late types for connection properties creation in the [Creation Late Types for Model Features](#) article.

Important : For connection features you will have to use "PRD1", "PRD2", etc... to set the different supports and "Handler0D" or "Handler1D" for fastener placements.

7. Setting the spring connection attributes

In this step UC retrieves the connector section of the spring and sets its elasticities.

```
...
' Retrieving the connector section

mySpringConnection.SpringType = SimGeneralSpring

Dim myConnectionSection As SimConnectorSection
Set myConnectionSection = mySpringConnection.ConnectorSection

' Retrieving the behavior

Dim myConnectionBehavior As SimConnectorBehavior
Set myConnectionBehavior = myConnectionSection.ConnectorBehavior

' Retrieving the elasticities

Dim myConnectorElasticities As SimConnectorElasticities
Set myConnectorElasticities = myConnectionBehavior.ConnectorElasticities
```

```

' Retrieving the first elasticity
Dim myConnectorElasticity1 As SimConnectorElasticity
Set myConnectorElasticity1 = myConnectorElasticities.Item(1)
myConnectorElasticity1.ElasticityOrder = SimConnectorElasticityNonLinear

' Retrieving the stiffness column
Dim myStiffnessColumn1 As SimTableColumn
Set myStiffnessColumn1 = myConnectorElasticity1.GetTableColumn(SimConnectorElasticityStiffness)

' Retrieving the position column
Dim myPositionColumn1 As SimTableColumn
Set myPositionColumn1 = myConnectorElasticity1.GetTableColumn(SimConnectorElasticityPosition)

' Setting data
myStiffnessColumn1.SetCellData 1, 0.17
myPositionColumn1.SetCellData 1, 0.008
...

```

Creating a Connection Property

This use case primarily focuses on the methodology to create a connection property.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\`, where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Launch the structural scenario App and create a structural simulation.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaConnectionsWithPythonSource.htm](#)

This use case can be divided into seven steps:

1. [Retrieves the simulation and its product](#)
2. [Retrieves the publications and composes links](#)
3. [Creates a rigid engineering connection](#)
4. [Creates constraints](#)
5. [Creates the FEM representation and retrieves its root object](#)
6. [Retrieves the MCX property manager](#)
7. [Creates a spring connection property](#)
8. [Sets the spring connection attributes](#)

1. Retrieves the simulation and its product

As a first step, the UC retrieves the simulation and its product from the simulation editor.

```

...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...

```

Then, opens the simulation model and retrieves the root occurrence and the root product

```

...
oOpenService = CATIA.GetService("PLMOpenService")
oOpenService.PLMOpen (myModel)

myProductEditor = CATIA.ActiveEditor
myProductService = myProductEditor.GetService("PLMProductService")

myProductEntities = myProductService.EditedContent
MyRootProduct = myProductEntities.Item(1)

myRootOccurrence = myProductService.RootOccurrence
...

```

2. Retrieves the publications and composing links

In this step UC retrieves the publications aggregated by the root product and composes links.

```

...
myPublications = MyRootProduct.Publications
myCoverContactFace= myPublications.GetItem("Cover_Contact_Face")
...

```

then, composes link to the publication

```

...
myLinkToCoverContactFace = myProductService.ComposeLink(myRootOccurrence, None, myCoverContactFace)
...

```

3. Creates a rigid engineering connection and creates constraints

In this step UC creates a rigid engineering connection.

First, Retrieve the engineering connection manager

```

...
myEngineeringConnections = MyRootProduct.GetItem("CATEngConnections")
...

Create the list of impacted product

...
myImpacted = []
myImpacted.append("ws3_cover_part.1")
myImpacted.append("ws3_pump_housing_part.1")
...

Create a rigid engineering connection

...
myNewRigidEngineeringConnection = myEngineeringConnections.Add(win32com.client.constants.catRigid, myImpacted)
...

```

4. Creates constraints

In this step UC creates constraints.

First, retrieve the assembly constraint manager

```

...
myAssemblyConstraints = myNewRigidEngineeringConnection.AssemblyConstraints
...

Create coincidence constraint between two hole axis

...
myGeometries = []
myGeometries.append("Cover_Hole2_Axis")
myGeometries.append("Pump_Housing_Hole2_Axis")

myCoincidenceConstraint1 = myAssemblyConstraints.Add(win32com.client.constants.catCoincidenceLineLine, myGeometries)
...

```

5. Creates the FEM representation and retrieves its root object

In this step UC creates a new FEM representation object. To perform the creation, a simulation representation factory has to be retrieved on the reference object on which the FEM rep will be aggregated.

```

...
MyPrdRepFactory = MyRootProduct.GetItem("SimPrdRepFactory")
MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")
MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")
...

```

6. Retrieves the MCX property manager

In this step UC retrieves the MCX property manager.

```

...
myMCXProperties = myNewRigidEngineeringConnection.GetItem("SimMCXProperties")
...

```

7. Creates a spring connection property

In this step UC creates a spring connection property.

```

...
mySpringConnection = myMCXProperties.Add("SimSpring", MyFemRepRoot)
...

```

Setting of the Tie connection property supports

```

...
myLinkAccess = mySpringConnection.GetItem("SimLinkAccess")
...

```

Adding links for master and slave surface

```

...
# PRD1 attribute is for master surface
myLinkAccess.AddLink ("PRD1", myLinkToCoverContactFace)

# PRD2 attribute is for slave surface
myLinkAccess.AddLink ("PRD2", myLinkToPumpHousingContactFace)
...

```

You can find the available late types for connection properties creation in the [Creation Late Types for Model Features](#) article.

Important : For connection features you will have to use "PRD1", "PRD2", etc... to set the different supports and "Handler0D" or "Handler1D" for fastener placements.

8. Sets the spring connection attributes

In this step UC retrieves the connector section of the spring and sets its elasticities.

```

...
# Retrieving the connector section
mySpringConnection.SpringType = win32com.client.constants.SimGeneralSpring

myConnectionSection = mySpringConnection.ConnectorSection

```

```

# Retrieving the behavior
myConnectionBehavior = myConnectionSection.ConnectorBehavior

# Retrieving the elasticities
myConnectorElasticities = myConnectionBehavior.ConnectorElasticities

# Retrieving the first elasticity
myConnectorElasticity1 = myConnectorElasticities.Item(1)

myConnectorElasticity1.ElasticityOrder = win32com.client.constants.SimConnectorElasticityNonLinear

# Retrieving the stiffness column
myStiffnessColumn1 = myConnectorElasticity1.GetTableColumn(win32com.client.constants.SimConnectorElasticityStiffness)

# Retrieving the position column
myPositionColumn1 = myConnectorElasticity1.GetTableColumn(win32com.client.constants.SimConnectorElasticityPosition)

# Setting data

myStiffnessColumn1.SetCellData (1, 0.17)
myPositionColumn1.SetCellData (1, 0.008)
...

```

Managing Connection Property Features

This use case primarily focuses on the methodology to include and exclude connection property into the FEM representation.

Before you begin:

- Launch CATIA and then import and open the CAAScdFeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaManagingConnectionPropertySource.htm](#)

This use case can be divided in seven steps:

1. [Retrieving the active product](#)
2. [Creating the FEM representation and retrieving its root object](#)
3. [Adding associated rep to the FEM rep](#)
4. [Creating meshes on the Cover and the Pump Housing](#)
5. [Creating and including a connection property](#)
6. [Checking the connection property inclusion](#)
7. [Updating the FEM representation](#)
8. [Excluding the connection property](#)
9. [Including the connection property again](#)

1. Retrieving the active product

As a first step, the UC retrieves the active product.

```

...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor

Dim myProdService As PLMProductService
Set myProdService = myEditor.GetService("PLMProductService")

Dim myEntities As PLMEntities
Set myEntities = myProdService.EditedContent

Dim myRootProduct As VPMReference
Set myRootProduct = myEntities.Item(1)
...

```

2. Creating the FEM representation and retrieving its root object

In this step UC creates a new FEM representation object. To perform the creation, a simulation representation factory have to be retrieved on the reference object on which the FEM rep will be aggregated.

```

...
Dim myPrdRepFactory As SimPrdRepFactory
Set myPrdRepFactory = myRootProduct.GetItem("SimPrdRepFactory")

Dim myFemRepRef As VPMRepReference
Set myFemRepRef = myPrdRepFactory.CreatePrdRep("FEM")

Dim myFemRepRoot As SimFemRoot
Set myFemRepRoot = myFemRepRef.GetItem("SimFemRoot")
...

```

3. Adding associated rep to the FEM rep

In this step UC associates 3D shape with the FEM representation.

For further information about managing the FEM associated representation refer to the "Associates the PartBody with the FEM representation" section of the article [Creating a FEM rep and managing its associated shapes](#).

4. Creating meshes on the Cover and the Pump Housing

In this step UC creates mesh parts on the cover and pump housing shapes.

For further information about creates mesh part refer to the "Creates a Mesh Part" section of the article [Creating Mesh Part](#).

5. Creating connection property

In this step UC creates connection property and includes it in the FEM representation upon addition.

For further information about creating a connection property refer to the "Creating a spring connection property" section of the article [Creating a Connection Property](#).

6. Checking the connection property inclusion

In this step UC checks whether the connection property is included or not.

```
...
myFemRepRoot.IsIncludedPropertyConnection myRootOccurrence, mySpringConnection ' Connection property is created upon addition
...
```

7. Updating the FEM representation

In this step UC updates the FEM representation with the previously included connection property.

```
...
myFemRepRoot.Update
...
```

8. Excluding the connection property

In this step UC excludes the connection property from the FEM representation.

```
...
myFemRepRoot.ExcludePropertyConnection myRootOccurrence, mySpringConnection
...
```

9. Including the connection property again

In this step UC includes the connection property in the FEM representation again.

```
...
myFemRepRoot.IncludePropertyConnection myRootOccurrence, mySpringConnection
...
```

Managing Connection Property Features

This use case primarily focuses on the methodology to include and exclude connection property into the FEM representation.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Launch the structural scenario App and create a structural simulation.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaManagingConnectionPropertyWithPythonSource.htm](#)

This use case can be divided in seven steps:

1. [Retrieves the simulation and its product](#)
2. [Creates the FEM representation and retrieves its root object](#)
3. [Creates a spring property connection](#)
4. [Checks the connection property inclusion](#)
5. [Updates the FEM representation](#)
6. [Excludes the connection property](#)
7. [Includes the connection property again](#)

1. Retrieves the simulation and its product

As a first step, the UC retrieves the simulation and its product from the simulation editor.

```
...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...
```

Then, opens the simulation model and retrieves the root occurrence and the root product

```
...
oOpenService = CATIA.GetService("PLMOpenService")
oOpenService.PLMOpen (myModel)

myProductEditor = CATIA.ActiveEditor
myProductService = myProductEditor.GetService("PLMProductService")

myProductEntities = myProductService.EditedContent
MyRootProduct = myProductEntities.Item(1)

myRootOccurrence = myProductService.RootOccurrence
...
```

2. Creates the FEM representation and retrieves its root object

In this step UC creates a new FEM representation object. To perform the creation, a simulation representation factory have to be retrieved on the reference object on which the FEM rep will be aggregated.

```

MyPrdRepFactory = MyRootProduct.GetItem("SimPrdRepFactory")
MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")
MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")
...

```

3. Creates connection property

In this step UC creates connection property and includes it in the FEM representation upon addition.

For further information about creating a connection property refer to the spring connection property section of the article [Creating a Connection Property](#).

4. Checks the connection property inclusion

In this step UC checks whether the connection property is included or not.

```

...
MyFemRepRoot.IsIncludedPropertyConnection (myRootOccurrence, mySpringConnection)
...

```

5. Updates the FEM representation

In this step UC updates the FEM representation with the previously included connection property.

```

...
MyFemRepRoot.Update
...

```

6. Excludes the connection property

In this step UC excludes the connection property from the FEM representation.

```

...
MyFemRepRoot.ExcludePropertyConnection (myRootOccurrence, mySpringConnection)
...

```

7. Includes the connection property again

In this step UC includes the connection property in the FEM representation again.

```

...
MyFemRepRoot.IncludePropertyConnection (myRootOccurrence, mySpringConnection)
...

```

Creating a Fastener Connection Property

This use case primarily focuses on the methodology to create a fastener connection property.

Before you begin:

- Launch CATIA and then import and open the CAAScdFeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- CA media is required in order to create and edit fasteners.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdFeaFastenersSource.htm](#)

This use case can be divided in seven steps:

1. [Retrieving the active product](#)
2. [Retrieving the publications and composing links](#)
3. [Creating and initializing a core material](#)
4. [Creating an engineering connection](#)
5. [Creating the FEM representation and retrieving its root object](#)
6. [Creating and initializing a line fastener connection property](#)
7. [Creating and initializing a point fastener connection property](#)

1. Retrieving the active product

As a first step, the UC retrieves the active product.

```

...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor

Dim myProdService As PLMProductService
Set myProdService = myEditor.GetService("PLMProductService")

Dim myEntities As PLMEntities
Set myEntities = myProdService.EditedContent

Dim myRootProduct As VPMReference
Set myRootProduct = myEntities.Item(1)
...

```

2. Retrieving the publications and composing links

In this step UC retrieves the publications aggregated by the root product and composes links.

```

...
Dim myPublications As VPMPublications
Set myPublications = myRootProduct.Publications

Dim myPumpHousing As VPMPublication
Set myPumpHousing = myPublications.GetItem("Pump_Housing")

```

```
...
' Composing link to the publication

Dim myRootOccurrence As VPMRootOccurrence
Set myRootOccurrence = myProdService.RootOccurrence

Dim myRepInstance As VPMRepInstance
Set myRepInstance = Nothing ' Because we use publications

Dim myLinkToPumpHousing As AnyObject
Set myLinkToPumpHousing = myProdService.ComposeLink(myRootOccurrence, myRepInstance, myPumpHousing)
...
```

3. Creating and initializing a core material

In this step UC creates a core material which will be applied on fastener properties.

For further information about creating a core material refer to the article [Creating and Modifying Simulation Material Domain](#).

4. Creating an engineering connection

In this step UC creates an engineering connection.

For further information about creating a rigid engineering connection refer to the article [Creating an Assembly Constraint Between Two Instances of a Basic Pad](#).

5. Creating the FEM representation and retrieving its root object

In this step UC creates a new FEM representation object. To perform the creation, a simulation representation factory have to be retrieved on the reference object on which the FEM rep will be aggregated.

```
...
Dim myPrdRepFactory As SimPrdRepFactory
Set myPrdRepFactory = myRootProduct.GetItem("SimPrdRepFactory")

Dim myFemRepRef As VPMRepReference
Set myFemRepRef = myPrdRepFactory.CreatePrdRep("FEM")

Dim myFemRepRoot As SimFemRoot
Set myFemRepRoot = myFemRepRef.GetItem("SimFemRoot")
...
```

6. Creating and initializing a line fastener connection property

In this step UC creates a line fastener connection property.

```
...
' Creating the line fastener connection property

Dim myMCXProperties As SimMCXProperties
Set myMCXProperties = myEngineeringConnection.GetItem("SimMCXProperties")

Dim myLineFastener As SimLineFastener
Set myLineFastener = myMCXProperties.Add("SimLineFastener", myFemRepRoot)

' Setting the line fastener connection property supports

Dim myLineFastenerLinkAccess As SimLinkAccess
Set myLineFastenerLinkAccess = myLineFastener.GetItem("SimLinkAccess")

myLineFastenerLinkAccess.AddLink "PRD1", myLinkToPumpHousing ' PRD1 attribute is for master surface
myLineFastenerLinkAccess.AddLink "PRD2", myLinkToCover ' PRD2 attribute is for slave surface
myLineFastenerLinkAccess.AddLink "Handler1D", myLinkToLine ' Handler1D attribute is for fasteners placements

' Setting the line fastener connection property material

Dim myLinkToLineFastener As AnyObject
Set myLinkToLineFastener = myProdService.ComposeLink(myRootOccurrence, Nothing, myLineFastener)

Dim myCoreAppliedMaterial As AppliedMaterial
myMaterialService.SetMaterialCore myLinkToLineFastener, myCoreMatRef, myCoreAppliedMaterial

' Setting the line fastener connection property attributes

myLineFastener.ConstructType = SimLineFastenerShell
myLineFastener.CouplingType = SimCouplingKinematic
myLineFastener.MeshCompatibility = SimLineFastenerNonCompatible
myLineFastener.Height = 0.001
myLineFastener.MaximumAngle = 2
myLineFastener.Spacing = 0.003
myLineFastener.Width = 0.004
...
```

7. Creating and initializing a point fastener connection property

In this step UC creates a point fastener connection property.

```
...
' Creating the point fastener connection property

Dim myPointFastener As SimPointFastener
Set myPointFastener = myMCXProperties.Add("SimPointFastener", myFemRepRoot)

' Setting the point fastener connection property supports

Dim myPointFastenerLinkAccess As SimLinkAccess
Set myPointFastenerLinkAccess = myPointFastener.GetItem("SimLinkAccess")

myPointFastenerLinkAccess.AddLink "PRD1", myLinkToPumpHousing ' PRD1 attribute is for master surface
myPointFastenerLinkAccess.AddLink "PRD2", myLinkToCover ' PRD2 attribute is for slave surface
```

```

myPointFastenerLinkAccess.AddLink "Handler0D", myLinkToLine ' Handler0D attribute is for fasteners placements
' Setting the point fastener connection property material
Dim myLinkToPointFastener As AnyObject
Set myLinkToPointFastener = myProdService.ComposeLink(myRootOccurrence, Nothing, myPointFastener)

myMaterialService.SetMaterialCore myLinkToPointFastener, myCoreMatRef, myCoreAppliedMaterial

' Setting the point fastener connection property attributes

myPointFastener.ConstructType = SimPointFastenerRigid
myPointFastener.CouplingType = SimCouplingKinematic
myPointFastener.FastenerDiameter = 0.002

Dim myPointFastenerPlacement As SimPointFastenerPlacement
Set myPointFastenerPlacement = myPointFastener.PointFastenerPlacement
myPointFastenerPlacement.FastenerPlacementMethod = SimPointFastenerPlacementLine
myPointFastenerPlacement.NumberOfFastenersAlongLines = 20
...

```

Creating a Fastener Connection Property

This use case primarily focuses on the methodology to create a fastener connection property.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Launch the structural scenario App and create a structural simulation.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaFastenersWithPythonSource.htm](#)

This use case can be divided in seven steps:

1. [Retrieves the simulation and its product](#)
2. [Retrieves the publications and composing links](#)
3. [Creates and initializes a core material](#)
4. [Creates an engineering connection](#)
5. [Creates the FEM representation and retrieves its root object](#)
6. [Creates and initializes a line fastener connection property](#)
7. [Creates and initializes a point fastener connection property](#)

1. Retrieves the simulation and its product

As a first step, the UC retrieves the active editor and retrieves the simulation model.

```

...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model

```

Then opens the simulation model and retrieves the root occurrence.

```

oOpenService = CATIA.GetSessionService("PLMOpenService")
oOpenService.PLMOpen (myModel)

myProductEditor = CATIA.ActiveEditor
MyProductService = myProductEditor.GetService("PLMProductService")

myEntities = MyProductService.EditedContent
myRootProduct = myEntities.Item(1)

myRootOccurrence = MyProductService.RootOccurrence
...

```

2. Retrieves the publications and composing links

In this step UC retrieves the publications aggregated by the root product and composes links.

```

...
myPublications = myRootProduct.Publications

myPumpHousing = myPublications.GetItem("Pump_Housing")

# Composing link to the publication
myLinkToPumpHousing = myProdService.ComposeLink(myRootOccurrence, None, myPumpHousing)
...

```

3. Creates and initializes a core material

In this step UC creates a core material which will be applied on fastener properties.

For further information about creating a core material refer to the article [Creating and Modifying Simulation Material Domain With Python](#).

4. Creates an engineering connection

In this step UC creates an engineering connection.

```

...
# Creating a free engineering connection
myEngineeringConnections = myRootProduct.GetItem("CATEngConnections")

```

```

# Create the list of impacted product
myImpacted = []
myImpacted.append("ws3_pump_housing_part.1")
myImpacted.append("ws3_cover_part.1")

# Create a rigid engineering connection
myEngineeringConnection = myEngineeringConnections.Add(win32com.client.constants.catFree, myImpacted)

...

```

5. Creates the FEM representation and retrieves its root object

In this step UC creates a new FEM representation object. To perform the creation, a simulation representation factory have to be retrieved on the reference object on which the FEM rep will be aggregated.

```

...
MyPrdRepFactory = myRootProduct.GetItem("SimPrdRepFactory")
MyFemRepRef = MyPrdRepFactory.CreatePrdRep("FEM")
MyFemRepRoot = MyFemRepRef.GetItem("SimFemRoot")
...

```

6. Creates and initializes a line fastener connection property

In this step UC creates a line fastener connection property.

```

...
# Creating the line fastener connection property
myMCXProperties = myEngineeringConnection.GetItem("SimMCXProperties")
myLineFastener = myMCXProperties.Add("SimLineFastener", MyFemRepRoot)

# Setting the line fastener connection property supports
myLineFastenerLinkAccess = myLineFastener.GetItem("SimLinkAccess")

# PRD1 attribute is for master surface
myLineFastenerLinkAccess.AddLink ("PRD1", myLinkToPumpHousing)

# PRD2 attribute is for slave surface
myLineFastenerLinkAccess.AddLink ("PRD2", myLinkToCover)

# Handler1D attribute is for fasteners placements
myLineFastenerLinkAccess.AddLink ("Handler1D", myLinkToLine)

# Setting the line fastener connection property material
myLinkToLineFastener = myProdService.ComposeLink(myRootOccurrence, None, myLineFastener)

myCoreAppliedMaterial = myMaterialService.SetMaterialCore (myLinkToLineFastener, myCoreMatRef, None)

# Setting the line fastener connection property attributes
myLineFastener.ConstructType = win32com.client.constants.SimLineFastenerShell
myLineFastener.CouplingType = win32com.client.constants.SimCouplingKinematic
myLineFastener.MeshCompatibility = win32com.client.constants.SimLineFastenerNonCompatible
myLineFastener.Height = 0.001
myLineFastener.MaximumAngle = 2
myLineFastener.Spacing = 0.003
myLineFastener.Width = 0.004
...

```

7. Creates and initializes a point fastener connection property

In this step UC creates a point fastener connection property.

```

...
# Creating the point fastener connection property
myPointFastener = myMCXProperties.Add("SimPointFastener", MyFemRepRoot)

# Setting the point fastener connection property supports
myPointFastenerLinkAccess = myPointFastener.GetItem("SimLinkAccess")

myPointFastenerLinkAccess.AddLink ("PRD1", myLinkToPumpHousing) # PRD1 attribute is for master surface
myPointFastenerLinkAccess.AddLink ("PRD2", myLinkToCover) # PRD2 attribute is for slave surface
myPointFastenerLinkAccess.AddLink ("Handler0D", myLinkToLine) # Handler0D attribute is for fasteners placements

# Setting the point fastener connection property material
myLinkToPointFastener = myProdService.ComposeLink(myRootOccurrence, None, myPointFastener)

myCoreAppliedMaterial = myMaterialService.SetMaterialCore (myLinkToPointFastener, myCoreMatRef, None)

# Setting the point fastener connection property attributes

myPointFastener.ConstructType = win32com.client.constants.SimPointFastenerRigid
myPointFastener.CouplingType = win32com.client.constants.SimCouplingKinematic
myPointFastener.FastenerDiameter = 0.002

myPointFastenerPlacement = myPointFastener.PointFastenerPlacement
myPointFastenerPlacement.FastenerPlacementMethod = win32com.client.constants.SimPointFastenerPlacementLine
myPointFastenerPlacement.NumberOfFastenersAlongLines = 20
...

```

Automated FEM Procedures

Technical Article

Abstract

This article is in addition to the one about Automated FEM tool in the product documentation. See [Simulation\Physics Simulation\Model Assembly Design\Automated Modeling](#)

It describes the available Automated FEM procedures to create mesh parts and it gives you some explanations to create your own user procedure.

How to test out of the box procedures

Out of the box procedures

User procedures

- [Procedure type](#)
- [Global variable](#)
- [Inputs](#)
- [Procedure syntax](#)

Example : Generate 3D mesh procedure

How to test out of the box procedures

For 3D procedures :

- Launch CATIA and import the `CAAScdFeaSample3DForAutomatedModeling.3dxml` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaModeling\samples\`, where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Open the "CAAScdFeaSample3DForAutomatedModeling" and launch the Model Assembly Design apps.
- Launch the Automated FEM tool in the Automated FEM section.

For 2D procedures :

- Launch CATIA and import the `CAAScdFeaSample2DForAutomatedModeling.3dxml` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaModeling\samples\`, where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Open the "CAAScdFeaSample2DForAutomatedModeling" and launch the Model Assembly Design apps.
- Launch the Automated FEM tool in the Automated FEM section.

Out of the box procedures

The table below describes the existing procedures that you can use.

There are two types of procedures :

- Rule based

Procedure name	Used Mesh part	Late type	Procedure
Generate3DMesh	Tetrahedron	CATFmtTetRulesMesher	CAAScdFeaGenerate3DMesh
Generate3DOctreeMesh	Octree 3D	CATFmtOctree3DRulesMesher	CAAScdFeaGenerate3DOctreeMesh
Generate3DSweepMesh	Sweep 3D	CATFmtSweep3DRulesMesher	CAAScdFeaGenerate3DSweepMesh
GenerateMidSurfWithAbsShape	Quadrangle Surface	CATFmtSurfQuadMesher	CAAScdFeaGenerateMidSurfWithAbsShape
GenerateSurfQuadMesh	Quadrangle Surface	CATFmtSurfQuadMesher	CAAScdFeaGenerateSurfQuadMesh
GenerateSurfTriMesh	Triangle Surface	CATFmtSurfTriMesher	CAAScdFeaGenerateSurfTriMesh

- Parameter based

Procedure name	Used Mesh part	Late type	Procedure
Generate3DOctreeMeshParameters	Octree 3D	CATFmtOctree3DRulesMesher	CAAScdFeaGenerate3DOctreeMeshParameters
Generate3DSweepMeshParameters	Sweep 3D	CATFmtSweep3DMesher	CAAScdFeaGenerate3DSweepMeshParameters
Generate3DTetMeshParameters	Tetrahedron	CATFmtTetRulesMesher	CAAScdFeaGenerate3DTetMeshParameters
GenerateMidSurfWithAbsShapeParameters	Quadrangle Surface	CATFmtSurfQuadMesher	CAAScdFeaGenerateMidSurfWithAbsShapeParameters
GenerateSurfQuadMeshParameters	Quadrangle Surface	CATFmtSurfQuadMesher	CAAScdFeaGenerateSurfQuadMeshParameters
GenerateSurfTriMeshParameters	Triangle Surface	CATFmtSurfTriMesher	CAAScdFeaGenerateSurfTriMeshParameters

User procedures

This section explains how to create User procedures.

The Automated FEM tool allows to mesh assemblies using existing procedures or created user procedures. When using user procedures you can automate many tasks in addition to the predefined meshing procedures.

User procedures must follow a specific format to be used within the automated fem application.

Procedure type

All the procedures are in VB scripts. The extension is ".catvbs"

Global variable

Two global variables definition, SHAPE and FEM, are available and can be used.

The global variable SHAPE refers to the 3D Shape selection in the Batch Modeling dialog box. It is used for scripting geometry operations, such as defeaturig or midsurface extraction.

The global variable FEM refers to the finite element model representation associated with a 3D Shape. It is used for finite element operations, such as creating meshes or property assignments for use in simulations.

Inputs

The input definition is used to create an UI with the described parameters and let the user set interactively the required parameters. These must be entered before the procedure can be run.

Inputs are written in HTML as a comment in vb script to let the Automated FEM tool generates each requested UI. And then, during the script execution, these

information are given as input arguments to the VB script

Input definition is optional for user procedures. You can directly give the needed information on the VB script to run directly without setting interactively the required parameters.

Procedure syntax

The syntax is the standard VB scripting one. All the steps must be in between Sub main and end sub.

When using Inputs definitions, you have to give the same number of argument as input to the Sub main (iInput_1, iInput_2, ... , iInput_n)

Otherwise, there is no input argument for the Sub main.

For more information see "About the Automated FEM Application" and "About User Procedures" in the product documentation (Simulation / Physics Simulation / Model Assembly Design / Automated Modeling)

Example : Generate 3D mesh procedure

This procedure can be divided in 2 parts

1. [Inputs definition](#)
2. [VB Script](#)

The script can be divided in six steps

1. [Retrieve created FEM Rep](#)
2. [Create Mesh Part](#)
3. [Create Solid Property](#)
4. [Retrieve Mesh Part Support if it is not given as argument](#)
5. [Add support to Mesh part](#)
6. [Set Rule to Mesh part](#)

1. **Inputs definition**

In this example, three inputs are defined : Support, Meshing Rule and Property.

Automated FEM tool generates a specific dialog box and lets the user evaluate each input.

```
'<Inputs>
'<Input Name = "Support" Type ="Port" Mand ="false" Nls ="Support"/>
'<Input Name = "MeshingRule" Type ="DmtDocument" SubType ="CATFmtTetRule" Mand ="true" Nls ="Rule"/>
'<Input Name = "Property" Type ="Boolean" Mand ="true" Nls ="CreateSolid"/>
'</Inputs>
```

For each input four arguments have been given.

Name The name of the input

Type The Type of the input

Mand This argument is a boolean. It has to be set as True if this input is mandatory. False in the other case

Nls Description name of the input

Important: This part does not set the inputs value. It just defines requested inputs.

2. **VB Script**

1. **Retrieve created FEM Rep**

The Fem rep is created before the script execution and stored on the FEM global variable.

We retrieve in this step the SimFemRoot object on the created FEM

```
...
Dim FEMRoot As CATIAFmtFemRoot
Set FEMRoot = FEM.GetItem("SimFemRoot")
...
```

2. **Create Mesh Part**

Then we create the rule based 3D Mesh

```
...
' Retrieve the mesh set
Dim MeshSet As CATIAFmtMeshSet
Set MeshSet = FEMRoot.GetSet("SimNodesElements")

' Retrieve the mesh parts collection
Dim MeshParts As CATIAFmtMeshParts
Set MeshParts = MeshSet.MeshParts

' Add a new Mesh part to the collection of mesh parts
Dim MeshPart As CATIAFmtMeshPart
Set MeshPart = MeshParts.Add("CATFmtTetRulesMesher")
...
```

3. **Create a Solid Property**

If requested, we then create a Solid property

```
...
' Retrieve the property set object SimProperties
Dim PropertySet As SimProperties
Set PropertySet = FEMRoot.GetSet("SimProperties")

' Add a Solid section to the property set
Dim SolidSection As SimSolidSection
```

```
Set SolidSection = PropertySet.Add("SMAMpaSolidProperty")
...
```

4. Retrieve Mesh Part Support

Then we retrieve the mesh part support if it is not given.

For that, we use the SHAPE global variable valuated before script execution.

```
...
' Retrieve the shape reference from the global variable SHAPE
Dim ShapeRef As VPMRepReference
Set ShapeRef = SHAPE.ReferenceInstanceOf

' Retrieve the part associated to the shape reference
Dim ThePart As Part
Set ThePart = ShapeRef.GetItem("Part")

' Retrieve the main body of the part
Dim PartBody As AnyObject
Set PartBody = ThePart.MainBody
...
```

5. Add support to Mesh part

when the support is explicitly defined, it has to be given directly as argument to the AddSupport method.

Otherwise create the link to add to the mesh part using the composeLink(iOccurrence, iShape, iRef) method of the SimComposeLinkServices object.

```
...
' Create the reference of the part body for the third argument
Dim Ref As Reference
Set Ref = ThePart.CreateReferenceFromObject(PartBody)

' First argument is not needed
Dim EmptyOcc As PLMOccurrence
Set EmptyOcc = Nothing

' Retrieve the SimComposeLinkServices and create the link associated to the part body
Dim FEMservice As SimComposeLinkServices
Set FEMservice = FEM.GetItem("SimComposeLinkServices")
Set Link = FEMservice.ComposeLink(EmptyOcc, SHAPE, Ref)

' Add link to mesh part
MeshPart.AddSupport (Link)

' Add the link to the solid section if requested
Dim LinkAccess As SimLinkAccess
Set LinkAccess = SolidSection.GetItem("SimLinkAccess")
LinkAccess.AddLink "MainSupport", Link
...
```

6. Set Rule to Mesh part

Then set the rule to the mesh part

```
...
MeshPart.SetRule iRule, "CATFmtTetRule"
...
```

Automated FEM User Methods

Technical Article

Abstract

This article describes how to write a User Method for Automated FEM App.

A user method is a XML file that describes the list of mesh and connection procedures to apply on several subsets of an assembly of products.

It defines a list of [assignments](#) applying a mesh or connection procedure on the result of a search ([selector](#)) with a set of input values ([inputValueSet](#)).

Description of tags and attributes

- [method](#)
 - [id](#)
 - [displayName](#)
- [selectorSet](#)
 - [selector](#)
 - [id](#)
 - [type](#)
 - [subType](#)
 - [string](#)
 - [comparator](#)
- [selectionSet](#)
 - [selection](#)
 - [valueName](#)
 - [part](#)
 - [id](#)
 - [occurrencePath](#)
 - [featurePath](#)

- [inputValueSet](#)
 - [inputValue](#)
 - [id](#)
 - [input](#)
 - [key](#)
 - [type](#)
 - [subType](#)
 - [value](#)
 - [structuration](#)
 - [type](#)
 - [generative](#)
 - [value](#)
 - [assignmentSet](#)
 - [assignment](#)
 - [type](#)
 - [id](#)
 - [searchDomain](#)
 - [procedure](#)
 - [inputs](#)

[Examples of mesh procedure inputs](#)

[Example of connection procedure inputs](#)

[Example of user method](#)

Description of tags and attributes

• [method](#)

Root tag which defines the method.

Example :

Mandatory: true <[method](#) DisplayName = "myMethod">

...

</[method](#)>

◦ [id](#)

type: string

Attribute which defines the internal name of the method.

Mandatory: false

◦ [displayName](#)

type: string

Attribute which defines the name of the method.

Mandatory: true

• [selectorSet](#)

Tag which contains the list of searches

Example :

Mandatory: true <[selectorSet](#)>

<[selector](#) id = "PartFor2D" type = "Attribute" subType = "V_Name" string = "CAE" comparator = "=">

</[selectorSet](#)>

◦ [selector](#)

Mandatory: true Tag which represents a query on the database and defines a list of parts on which an assignment is applied.

◦ [id](#)

type: string

Attribute which defines the name of the search.

Mandatory: true

◦ [type](#)

Attribute which defines the type of the search.

type: string Available values:

Mandatory: true

- "All": Query all the parts under the root product.

- "Visible": Query all the visible parts or shapes under the root product.

- "Attribute": Query products on a PLM attribute, the attribute is defined by the subType attribute.

◦ Required attribute : [subType](#)

- "MeshedSupport": Query meshed geometries.

- "**Publication**": Query products on the first publication fitting the string according to the comparator.
 - Required attribute : [string](#)
 - "**Path**": (Only for connection procedure) Query products, parts or geometries based on their paths in the product structure.
 - Required attribute : [string](#)
- subType
- type: string** If the type is "Attribute" it defines the name of the PLM attribute.
- Mandatory: true** if type="Attribute" **Example :** V_NAME
- string
- If the type is "Publication" it defines the substring of the publication name.
- If the type is "Attribute" it defines the value of the PLM attribute.
- If the type is "Path" it defines the search path that products, parts or geometries must comply with. Path in the product structure is built from the titles of product or part instances (the title of product reference in case of root product) and names of geometrical sets or features.
- type: string**
 - the product path to the part or product with "!" separating each object in the path. If the product path is not defined, search is performed on all the parts and products.
 - adding "!" and the geometrical path to the geometrical set or feature with "!" separating each object in the geometrical path. The first object must be a geometrical set. If the first object is not defined, geometrical features are searched in all geometrical sets.
- Mandatory: true** if type="Attribute", "Publication" or "Path"
- Example :**
- Product1!Product2!Part3
 - Product1!Part2|GeomSet1!GeomFeat2
 - |GeomSet1!GeomFeat2
 - !GeomFeat2
- comparator
- If the type is "Attribute" or "Publication"
- type: string** Available values:
- Mandatory: false**
 - "=": The attribute value or the publication name must contain the string attribute
 - "!=": The attribute value or the publication name must not contain the string attribute
- **selectionSet**
- Tag which contains the list of selections
- Example :**
- ```
<selectionSet>
```
- Mandatory: false** <[selection](#) valueName = "PointPlacementSelection"/>
- ```
<part id = "1" occurrencePath = "DesignSolution_Body.1!Fasteners.1" featurePath = "Point_Fasteners 1"/>
</selection>
</selectionSet>
```
- **selection**
- Mandatory: false** This tag is needed in case the type of an input is "Selection".
- **valueName**
- type: string** Attribute which defines the name of the selection.
- Mandatory: true**
- **part**
- Mandatory: true** Tag which defines an item of the selection.
- **id**
- type: string** Attribute which defines the id of the part.
- Mandatory: true**
- **occurrencePath**
- type: string** Attribute which defines the product path to the part or product with "!" separating each object in the path. If the product path is not defined, search is performed on all the parts and products.
- Mandatory: true** **Remark:** the Root Product is defined by the reference name and the other products and parts by the instance name.
- Example :** Product1!Product2!Part3
- **featurePath**
- type: string** Attribute which defines the geometrical path to the geometrical set or feature with "!" separating each object in the geometrical path. The first object must be a geometrical set. If the first object is not defined, geometrical features are searched in all geometrical sets.

<p>true</p> <p>Example : GeomSet1!GeomFeat2 or !GeomFeat2</p> <ul style="list-style-type: none"> ○ inputValueSet 																														
<p>Tag which contains the list of inputs.</p> <p>Example :</p> <pre><inputValueSet> <inputValue id = "inputValueSet1"> <input key = "SelectionMode" type = "List" subType = "SelectionDefinition" value = "1"/> <input key = "Support" type = "Port" value = "" /> <input key = "MeshingRule" type = "DmtDocument" subType = "CATFmtTetRule" value = "3D_Rule"/> <input key = "Property" type = "Boolean" value = "TRUE"/> </inputValue> </inputValueSet></pre>																														
<ul style="list-style-type: none"> ■ inputValue <p>Tag which defines a set of parameters needed as input for mesh or connection procedures.</p> <p>A procedure needs a specific list of inputs (more details in Automated FEM Procedures).</p> <p>An example of inputs list for some out-of-the-box mesh procedures can be find below in Examples of mesh procedure inputs.</p> <p>Mandatory: true</p> <p>Each input only requires from this specific list the key, the type (eventually the subType) and the value.</p> <p>There is no need to specify the attributes Mand, Nls, Min, Max, MinExcl, MaxExcl.</p> <p>The inputs must be written in the same order as given in the mesh procedures.</p> <ul style="list-style-type: none"> ■ id <p>type: string</p> <p>Attribute which defines the name used by the assignment.</p> <p>Mandatory: true</p> <ul style="list-style-type: none"> ■ input <p>Mandatory: true Tag which defines an input for mesh or connection procedures.</p> <ul style="list-style-type: none"> ■ key <p>type: string</p> <p>Attribute which defines the name of the input.</p> <p>Mandatory: true</p> <ul style="list-style-type: none"> ■ type <p>Attribute which defines the type of the input. For more information, see the Simulation Physics Simulation Model Assembly Design Automated FEM About User Procedures section of the User Assistance</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding-bottom: 5px;">type</th> <th style="text-align: left; padding-bottom: 5px;">subType</th> <th style="text-align: left; padding-bottom: 5px;">value</th> </tr> </thead> <tbody> <tr> <td style="padding-bottom: 5px;">Integer, Double, String, Boolean</td> <td style="padding-bottom: 5px;">-</td> <td style="padding-bottom: 5px;">integer, double, string, boolean value</td> </tr> <tr> <td style="padding-bottom: 5px;">DmtDocument</td> <td style="padding-bottom: 5px;">extension of the dmt document</td> <td style="padding-bottom: 5px;">name of the dmtdocument (meshing rule, connection rule, ...)</td> </tr> <tr> <td style="padding-bottom: 5px;">List</td> <td style="padding-bottom: 5px;">name of the list in the inputs definition</td> <td style="padding-bottom: 5px;">integer value (starting at 1) giving the position in the list</td> </tr> <tr> <td style="padding-bottom: 5px;">Enum</td> <td style="padding-bottom: 5px;">name of the enum in the inputs definition</td> <td style="padding-bottom: 5px;">integer value (starting at 1) giving the position in the enum</td> </tr> <tr> <td style="padding-bottom: 5px;">Mandatory: true</td> <td style="padding-bottom: 5px;">Dimension</td> <td style="padding-bottom: 5px;">literal value containing units ("10mm")</td> </tr> <tr> <td style="padding-bottom: 5px;"></td> <td style="padding-bottom: 5px;">DimensionRange</td> <td style="padding-bottom: 5px;">min/max values separated by ("0mm 20mm")</td> </tr> <tr> <td style="padding-bottom: 5px;"></td> <td style="padding-bottom: 5px;">Port</td> <td style="padding-bottom: 5px;">string value giving the name of publication</td> </tr> <tr> <td style="padding-bottom: 5px;">Selection</td> <td style="padding-bottom: 5px;">selection type (MonoSelectionSolids, MultiSelectionSolids, MultiSelectionLinePlacement,...)</td> <td style="padding-bottom: 5px;">string value giving the name of the selection defined in selectionSet</td> </tr> <tr> <td style="padding-bottom: 5px;">Search</td> <td style="padding-bottom: 5px;">-</td> <td style="padding-bottom: 5px;">string value giving the name of the search defined in selectorSet</td> </tr> </tbody> </table> <ul style="list-style-type: none"> ■ subType <p>type: string</p> <p>Attribute which defines the subtype of the input depending on the type.</p> <p>Mandatory: false Example : SelectionDefinition (List type), Length (Dimension type), ElementOrder (Enum)</p> <ul style="list-style-type: none"> ■ value <p>type: string</p> <p>Attribute which defines the value of the input depending on the type.</p> <p>Mandatory: true</p> <ul style="list-style-type: none"> ○ structuration 	type	subType	value	Integer, Double, String, Boolean	-	integer, double, string, boolean value	DmtDocument	extension of the dmt document	name of the dmtdocument (meshing rule, connection rule, ...)	List	name of the list in the inputs definition	integer value (starting at 1) giving the position in the list	Enum	name of the enum in the inputs definition	integer value (starting at 1) giving the position in the enum	Mandatory: true	Dimension	literal value containing units ("10mm")		DimensionRange	min/max values separated by ("0mm 20mm")		Port	string value giving the name of publication	Selection	selection type (MonoSelectionSolids, MultiSelectionSolids, MultiSelectionLinePlacement,...)	string value giving the name of the selection defined in selectionSet	Search	-	string value giving the name of the search defined in selectorSet
type	subType	value																												
Integer, Double, String, Boolean	-	integer, double, string, boolean value																												
DmtDocument	extension of the dmt document	name of the dmtdocument (meshing rule, connection rule, ...)																												
List	name of the list in the inputs definition	integer value (starting at 1) giving the position in the list																												
Enum	name of the enum in the inputs definition	integer value (starting at 1) giving the position in the enum																												
Mandatory: true	Dimension	literal value containing units ("10mm")																												
	DimensionRange	min/max values separated by ("0mm 20mm")																												
	Port	string value giving the name of publication																												
Selection	selection type (MonoSelectionSolids, MultiSelectionSolids, MultiSelectionLinePlacement,...)	string value giving the name of the selection defined in selectionSet																												
Search	-	string value giving the name of the search defined in selectorSet																												

Mandatory: false Tag which defines the AOM/MOA mode.

default value: "AOM"

- **type** Attribute which defines the type of structuration.

type: string Available values:

Mandatory: false

- "AOM": Create one FEM per part
- "MOA": Create one FEM containing all the meshes

- **generative**

Mandatory: Tag which defines the storage of connection procedures and results in an automated specification.

false The supported connection procedures are: Read From Design procedure with "In current finite element model" as Creation location, and the other procedures with "In current finite element model" as Creation location and applied on "Meshed Supports".

default value: "false" It enables user to relaunch connection procedures according to design and model changes.

- **value** Attribute which defines the generative storage.

type: string Available values:

Mandatory: false

- "true"
- "false"

- **assignmentSet** Tag which contains the list of assignments.

Example :

```
<assignmentSet>
  <assignment type = "Mesh" id = "1" searchDomain = "PartFor2D" procedure = "GenerateSurfQuadMesh" inputs = "inputValueSet1" />
  <assignment type = "Mesh" id = "2" searchDomain = "PartFor3D" procedure = "Generate3DMesh" inputs = "inputValueSet2" />
</assignmentSet>
```

- **assignment**

Mandatory: true Tag which defines an assignment. An assignment applies a mesh or connection procedure on the result of a search with a set of input values.

- **type** Attribute which defines the type of the assignment.

type: string Available values:

Mandatory: true

- "Mesh": Applies a mesh procedure
- "Connection": Applies a connection procedure

- **id** Attribute which defines the id of the assignment.

type: string Attribute which defines the name of the selector.

Mandatory: true

- **searchDomain** Attribute which defines the name of the selector.

type: string Attribute which defines the name of the procedure. This procedure contains the instructions to populate the FEM.

Example for mesh procedure: "Generate3DTetMeshParameters", "Ignore"...

Mandatory: true

Example for connection procedure: "GeometricalDetection", "GeometricalLineDetection", "GeometricalPointDetection", "ImportFromFSR".

- **inputs** Attribute which defines the name of the inputs set.

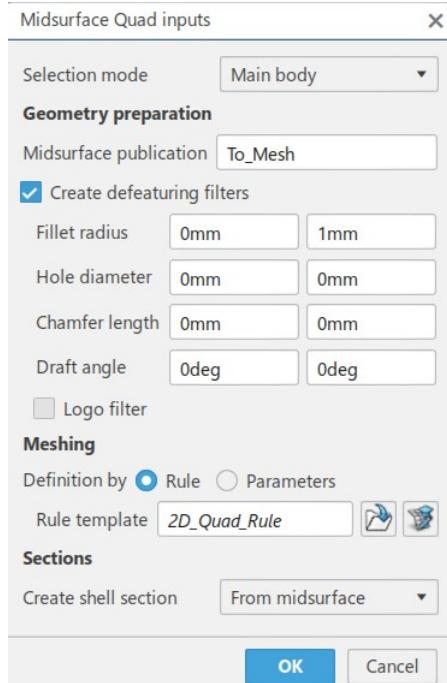
Mandatory: true

Examples of mesh procedure inputs

Here are some examples of inputs for the out-of-the-box mesh procedures available in Automated FEM.

Mesh procedure UI

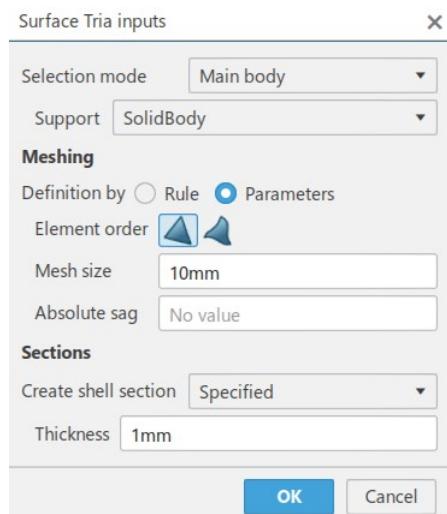
List of inputs



```

<inputValue id = "MidSurface_Rule_Based">
<input key = "SelectionMode" type = "List" subType = "SelectionDefinition"
      <input key = "PublicationName" type = "String" value = "To_Mes"
            <input key = "Defeaturing" type = "Boolean" value = "TRUE">
<input key = "FilletRadius" type = "DimensionRange" subType = "Length" va
<input key = "HoleDiameter" type = "DimensionRange" subType = "Length" va
<input key = "ChamferLength" type = "DimensionRange" subType = "Length" va
<input key = "DraftAngle" type = "DimensionRange" subType = "Angle" value
      <input key = "LogoFilter" type = "Boolean" value = "FALSE">
<input key = "MeshDefinition" type = "Enum" subType = "MeshDefinition"
<input key = "MeshingRule" type = "DmtDocument" subType = "CATFmtSurfQuadRule" v
      <input key = "MeshSize" type = "Dimension" subType = "Length" val
      <input key = "ElementOrder" type = "Enum" subType = "ElementOrder" v
            </input>
<input key = "ShellSection" type = "List" subType = "ThicknessDefinition"
      <input key = "Specified" type = "Dimension" subType = "Length" val
            </input>
</inputValue>

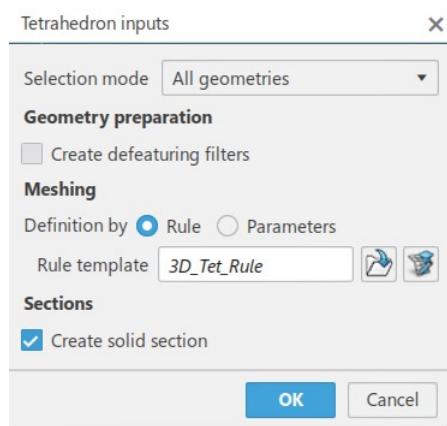
```



```

<inputValue id = "SurfTria_Parameters_Based">
<input key = "SelectionMode" type = "List" subType = "SelectionDefinition"
      <input key = "Support" port = "Boolean" value = "SolidBody"
            </input>
<input key = "MeshDefinition" type = "Enum" subType = "MeshDefinition"
      <input key = "MeshingRule" type = "DmtDocument" subType = "" val
      <input key = "MeshSize" type = "Dimension" subType = "Length" value
      <input key = "ElementOrder" type = "Enum" subType = "ElementOrder" v
            </input>
<input key = "ShellSection" type = "List" subType = "ThicknessDefinition"
      <input key = "Specified" type = "Dimension" subType = "Length" val
            </input>
</inputValue>

```



```

<inputValue id = "3DTet_Rule_Based">
<input key = "SelectionMode" type = "List" subType = "SelectionDefinition"
      <input key = "Support" port = "Boolean" value = ""/>
            </input>
      <input key = "Defeaturing" type = "Boolean" value = "FALSE">
<input key = "FilletRadius" type = "DimensionRange" subType = "Length" va
<input key = "HoleDiameter" type = "DimensionRange" subType = "Length" va
<input key = "ChamferLength" type = "DimensionRange" subType = "Length" va
<input key = "DraftAngle" type = "DimensionRange" subType = "Angle" value
      <input key = "LogoFilter" type = "Boolean" value = "FALSE">
<input key = "MeshDefinition" type = "Enum" subType = "MeshDefinition"
      <input key = "MeshingRule" type = "DmtDocument" subType = "" val
      <input key = "MeshSize" type = "Dimension" subType = "Length" value
      <input key = "ElementOrder" type = "Enum" subType = "ElementOrder" v
            </input>
<input key = "ShellSection" type = "List" subType = "ThicknessDefinition"
      <input key = "Specified" type = "Dimension" subType = "Length" val
            </input>
</inputValue>

```

Remarks:

The input list of `subType = "SelectionDefinition"` can take 3 integer values:

- o "1": for "MainBody".
- o "2": for "AllGeometries".
- o "2": for "VisibleGeometries".

The input list for `key = "ElementOrder"` can take 2 integer values:

- o "1": for "Linear".
- o "2": for "Quadratic".

The input list for `key = "ThicknessDefinition"` can take 4 integer values:

- "1": for "No shell section".
- "2": for "Specified".
- "3": for "From thin part feature".
- "4": for "From midsurface".

Example of connection procedure inputs

Here are some examples of inputs for each type of available connection procedures.

o Geometrical Point Detection

```
<inputValue id = "UserConnInputs2">
<input key = "PointRule" type = "DmtDocument" subType = "SMAFeaPointFastenerRule" value = "PF" revision = "---.000" />
<input key = "PointPlacement" type = "List" subType = "PlacementDefinition" value = "1"/>
<input key = "PointPlacementSearch" type = "Search" value = "UserInputSearch2_1"/>
<input key = "PointPlacementSelection" type = "Selection" subType = "MultiSelectionPointPlacement" value = ""/>
</input>
<input key = "Tolerance" type = "Dimension" subType = "Length" value = "15mm"/>
<input key = "Prefix" type = "String" value = "PF_"/>
<input key = "CreationLocation" type = "List" subType = "" value = "2"/>
</inputValue>
<selectorSet>
  <selector id = "UserInputSearch2_1" type = "Attribute" subType = "V_Name" string = "|Point" comparator = "="/>
</selectorSet>
```

o Geometrical Line Detection

```
<inputValue id = "UserConnInputs3">
<input key = "PointRule" type = "DmtDocument" subType = "SMAFeaLineFastenerRule" value = "Seam Weld (NVH)" revision = "---.000" />
<input key = "PlacementMethod" type = "List" subType = "PlacementMethodDefinition" value = "1"/>
<input key = "LinePlacement" type = "List" subType = "PlacementDefinition" value = "2"/>
<input key = "LinePlacementSearch" type = "Search" value = "UserInputSearch3_1"/>
<input key = "LinePlacementSelection" type = "Selection" subType = "MultiSelectionLinePlacement" value = "Selection1_3"/>
</input>
<input key = "Tolerance" type = "Dimension" subType = "Length" value = "10mm"/>
<input key = "Prefix" type = "String" value = "LF_"/>
<input key = "CreationLocation" type = "List" subType = "" value = "3"/>
</inputValue>
<selectionSet>
  <selection valueName = "Selection1_3">
    <part id = "1" occurrencePath = "#NDIVehicleReference ---.000!DesignSolution_Body.1!Fasteners.1" featurePath = "Adhesives">
    </selection>
  </selectionSet>
```

o Read From Design

```
<inputValue id = "UserConnInputs1">
<input key = "PointRule" type = "DmtDocument" subType = "SMAFeaPointFastenerRule" value = "PF" revision = "---.000" />
<input key = "Prefix" type = "String" value = "FSR_"/>
<input key = "CreationLocation" type = "List" subType = "" value = "1"/>
```

o Virtual Bolt Detection

```
<inputValue id = "UserConnInputs4">
<input key = "Geometry" type = "List" value = "1"/>
<input key = "PlacementMethod" type = "List" subType = "PlacementMethodDefinition" value = "1"/>
<input key = "PointPlacement" type = "List" subType = "PlacementDefinition" value = "1"/>
<input key = "PointPlacementSearch" type = "Search" value = "UserInputSearch1_1"/>
<input key = "PointPlacementSelection" type = "Selection" subType = "MultiSelectionPointPlacement" value = ""/>
</input>
<input key = "PointPlacementTolerance" type = "Dimension" subType = "Length" value = ""/>
<input key = "LinePlacement" type = "List" subType = "PlacementDefinition" value = "1"/>
<input key = "LinePlacementSearch" type = "Search" value = "UserInputSearch1_2"/>
<input key = "LinePlacementSelection" type = "Selection" subType = "MultiSelectionLinePlacement" value = ""/>
</input>
<input key = "LinePlacementTolerance" type = "Dimension" subType = "Length" value = ""/>
</input>
<input key = "CircularShape" type = "Boolean" value = "false"/>
<input key = "Diameter" type = "DimensionRange" subType = "Length" value = "0mm|50mm"/>
<input key = "Width" type = "DimensionRange" subType = "Length" value = "10mm|60mm"/>
<input key = "DiameterTolerance" type = "Dimension" subType = "Length" value = "2mm"/>
<input key = "Length" type = "DimensionRange" subType = "Length" value = "20mm|70mm"/>
<input key = "MaxOffsetBetweenHoles" type = "Dimension" subType = "Length" value = "5mm"/>
<input key = "MaxAngleBetweenAxes" type = "Dimension" subType = "Angle" value = "10deg"/>
<input key = "VirtualBoltRule" type = "DmtDocument" subType = "SMAFeaVirtualBoltRule" value = "SMAFeaVirtualBoltRule" revision = "---.000" />
<input key = "Prefix" type = "String" value = "Test_"/>
<input key = "CreationLocation" type = "List" subType = "" value = "4"/>
</inputValue>
<selectorSet>
  <selector id = "UserInputSearch1_1" type = "Path" string = "" comparator = "="/>
  <selector id = "UserInputSearch1_2" type = "Path" string = "" comparator = "="/>
</selectorSet>
```

Remarks:

The input list of `subType = "PlacementDefinition"` can take 2 integer values:

- "1": for "PointPlacementSearch" or "LinePlacementSearch".
- "2": for "PointPlacementSelection" or "LinePlacementSelection".

The input list for `key = "CreationLocation"` can take 4 integer values:

- "1": for "One engineering connection for all connection properties".
- "2": for "One engineering connection per connected parts".
- "3": for "One engineering connection per connection property".
- "4": for "In current finite element model".

The input list of `subType = "PlacementMethodDefinition"` (for line fasteners) can take 2 integer values:

- "1": for "Line".
- "2": for "Support boundary".

The input list of `subType = "PlacementMethodDefinition"` (for virtual bolts) can take 3 integer values:

- "1": for "Free"; no Placement block is required.
- "2": for "Point": the first Placement block is required.
- "3": for "Line": the second Placement block is required.

Tolerance value is set as maximum projection distance of detected connections, instead of maximum projection distance of connection rule.

Example of user method

Here is an example of a user method containing two mesh assignments and one connection assignment.

```
<?xml version='1.0' encoding='UTF-8' ?>
<method displayName = "ODTmethod">
  <selectorSet>
    <selector id = "PartFor2D" type = "Attribute" subType = "V_Name" string = "CAE" comparator = "="/>
    <selector id = "PartFor3D" type = "Attribute" subType = "V_Name" string = "NDISeat" comparator = "="/>
    <selector id = "ConnecSupport" type = "MeshedSupport"/>
  </selectorSet>

  <selectionSet>
    <selection valueName = "Selection1_1">
      <part id = "1" occurrencePath = "L8D2-5K091-A-PIA-01_51 ---.000!L8D2-5K091-A-PIA-26_12.1" featurePath = "ON_SOLIDS"/>
    </selection>
  </selectionSet>

  <inputValueSet>
    <inputValue id = "inputvalueSet1">
      <input key = "SelectionMode" type = "List" subType = "SelectionDefinition" value = "1">
        <input key = "Support" type = "Port" value = "To_Mesh" />
      </input>
      <input key = "MeshDefinition" type = "Enum" subType = "MeshDefinition" value = "1" >
        <input key = "MeshingRule" type = "DmtDocument" subType = "CATFmtSurfQuadRule" value = "2D_Quad_Rule"/>
        <input key = "MeshSize" type = "Dimension" subType = "Length" value = ""/>
        <input key = "ElementOrder" type = "Enum" subType = "ElementOrder" value = ""/>
      </input>
      <input key = "ShellSection" type = "List" subType = "ThicknessDefinition" value = "3">
        <input key = "Specified" type = "Dimension" subType = "Length" visible = "2" value = ""/>
      </input>
    </inputValue>
    <inputValue id = "inputvalueSet2">
      <input key = "SelectionMode" type = "List" subType = "SelectionDefinition" value = "1">
        <input key = "Support" type = "Port" value = "" />
      </input>
      <input key = "MeshDefinition" type = "Enum" subType = "MeshDefinition" value = "1" >
        <input key = "MeshingRule" type = "DmtDocument" subType = "CATFmtTetRule" value = "3D_Rule"/>
        <input key = "MeshSize" type = "Dimension" subType = "Length" value = ""/>
        <input key = "ElementOrder" type = "Enum" subType = "ElementOrder" value = ""/>
      </input>
      <input key = "Property" type = "Boolean" value = "TRUE"/>
    </inputValue>
    <inputValue id = "UserConnInputs1">
      <input key = "LineRule" type = "DmtDocument" subType = "SMAFeaLineFastenerRule" value = "Seam Weld (NVH)" revision = "---.00" />
      <input key = "PlacementMethod" type = "List" subType = "PlacementMethodDefinition" value = "2"/>
      <input key = "LinePlacement" type = "List" subType = "PlacementDefinition" value = "2">
        <input key = "LinePlacementSearch" type = "Search" value = "UserInputSearch1_1"/>
        <input key = "LinePlacementSelection" type = "Selection" subType = "MultiSelectionLinePlacement" value = "Selection1_1"/>
      </input>
      <input key = "Tolerance" type = "Dimension" subType = "Length" value = "10mm"/>
      <input key = "Prefix" type = "String" value = "DETECTED_"/>
      <input key = "CreationLocation" type = "List" subType = "" value = "2"/>
    </inputValue>
  </inputValueSet>

  <structuration type = "AOM"/>

  <assignmentSet>
    <assignment type = "Mesh" id = "1" searchDomain = "PartFor2D" procedure = "GenerateSurfQuadMesh" inputs = "inputvalueSet1" />
    <assignment type = "Mesh" id = "2" searchDomain = "PartFor3D" procedure = "Generate3DMesh" inputs = "inputvalueSet2"/>
    <assignment type = "Connection" id = "MyLineConnection" searchDomain = "ConnecSupport" procedure = "GeometricalDetection" />
  </assignmentSet>
</method>
```

Connection Services

This article describe how to use available connection services through the object `SimConnectionServices`, used to create automatic connections. It also provides use cases for each method.

Example :

- Given a `SimFemRoot` object you can retrieve a `SimConnectionServices` object as following:

```
Dim myFEMRepRoot As SimFemRoot
```

```
...
```

```
Dim myConnectionServices As SimConnectionServices
```

```
Set myConnectionServices = myFEMRepRoot.GetItem("SimConnectionServices")
```

Important : The FEM representation used for each method is the one on which the SimConnectionServices object is retrieved.

Before you begin :

- Launch CATIA and then import and open the model corresponding to each method. These files are supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdFeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Four methods are provided to create fastener connections

1. [ImportFromFSR](#)

Creates point fasteners from FSR products.

2. [ImportFromXML](#)

Creates fasteners connections from an XML file import containing all necessary data.

3. [CreatePointsFromSearch](#)

Automatically creates point fasteners.

4. [CreateLinesFromSearch](#)

Automatically creates Line fasteners.

ImportFromFSR

Sub **ImportFromFSR** (CATBSTR iPrefix)

Creates points fastener from importing FSR Products.

- **Parameter :**

[in] iPrefix : Prefix for the name of all created fastener connections.

Important : The method searches for FSR products on the root product associated to the FEM representation.

Example : For this example, import CAAScdFeaImportFromFSRModel.3DXml.

Important : Make sure to open the model with its FEM Reps before launching the script.

Where to find the macro: [CAAScdFeaImportFromFSRSource.htm](#)

This use case is divided in 8 steps.

1. Opens the assembly product
2. Retrieves the active editor
3. Retrieves the root occurrence
4. Retrieves the active product
5. Retrieves the rep instance
6. Retrieves the FEM Representation
7. Retrieves the connection services object

```
...
Dim myConnectionService As SimConnectionServices
Set myConnectionService = MyFemRepRoot.GetItem("SimConnectionServices")
```

...

8. Import point fastener from FSR product

...

```
myConnectionService.ImportFromFSR "Test_"
```

...

ImportFromXML

Sub **ImportFromXML** (CATBSTR iXMLFilePath, CATBSTR iPrefix)

Creates fastener connections from an XML file import containing all necessary data.

- **Parameters :**

[in] iXMLFilePath : Full path of the XML file to import.

[in] iPrefix : Prefix for the name of all created fastener connections.

Example : For this example, import CAAScdFeaXMLImportModel.3DXml.

Important : Make sure to open the model with its FEM Reps before launching the script.

You should also copy into the folder pointed by the "TEMP" environment variable the "importTest.xml.txt" file supplied in the same folder as the 3dxml file. The file extension must be changed to "xml" instead of "txt"

Where to find the macro: [CAAScdFeaImportFromXMLSource.htm](#)

This use case is divided in 8 steps.

1. Opens the assembly product
2. Retrieves the active editor
3. Retrieves the root occurrence
4. Retrieves the active product
5. Retrieves the rep instance
6. Retrieves the FEM Representation
7. Retrieves the connection services object

...

```
Dim myConnectionService As SimConnectionServices
Set myConnectionService = MyFemRepRoot.GetItem("SimConnectionServices")
```

...

8. Import fastener connection From XML file import

...

```
Dim sTempPath As String
sTempPath = CATIA.SystemService.Environ("TEMP")

Dim sXMLPath As String
sXMLPath = sTempPath & "/" & "importTest.xml"

myConnectionService.ImportFromXML sXMLPath, "Test_"

...
```

CreatePointsFromSearch

Sub **CreatePointsFromSearch** (CATSafeArrayVariant iHandlers, CATBSTR iPrefix, CATVariant iTolerance)

Automatically creates point fasteners by giving an handlers list containing point.

- **Parameters :**
- [in] iHandlers : List of points or geometrical set containing the candidates points.
- [in] iPrefix : Prefix for the name of all created fastener connections.
- [in] iTolerance : Double value used to limit the projection of the point on the supports.

Important : Points are retrieved on the given iHandlers. The search domain is limited to meshed parts supports (automatically retrieved using the FEM representation).

Example : For this example, import CAAcdFeaFastenerImportModel.3DXml.

Important : Make sure to open the model with its FEM Reps before launching the script.

Where to find the macro: [CAAcdFeaCreatePointsFromSearchSource.htm](#)

This use case is divided in 9 steps.

1. Opens the assembly product
2. Retrieves the active editor
3. Retrieves the root occurrence
4. Retrieves the active product
5. Retrieves the rep instance
6. Retrieves the FEM Representation and the part
7. Retrieves the connection services object

...

```
Dim myConnectionService As SimConnectionServices
Set myConnectionService = MyFemRepRoot.GetItem("SimConnectionServices")
```

...

8. Get the geometrical set named : Weld Edges for CAE

...

```
Dim hander(0)
Set hander(0) = MyPart.FindObjectByName("Weld Edges for CAE")
```

...

9. Creates point fasteners from search

...

```
myConnectionService.CreatePointsFromSearch hander, "Test_", 10.0
...
```

CreateLinesFromSearch

Sub **CreateLinesFromSearch** (CATSafeArrayVariant iHandlers, CATBSTR iPrefix, CATVariant iTolerance)

Automatically creates line fasteners by giving an handlers list containing lines.

- **Parameters :**
- [in] iHandlers : List of lines or geometrical set containing the candidates lines.
- [in] iPrefix : Prefix for the name of all created fastener connections.
- [in] iTolerance : Double value used to limit the projection of the line on the supports.

Important : Lines are retrieved on the given iHandlers. The search domain is limited to meshed parts supports (automatically retrieved using the FEM representation).

Example : For this example, import CAAcdFeaFastenerImportModel.3DXml.

Important : Make sure to open the model with its FEM Reps before launching the script.

Where to find the macro: [CAAscdFeaCreateLinesFromSearchSource.htm](#)

This use case is divided in 9 steps.

1. Opens the assembly product
 2. Retrieves the active editor
 3. Retrieves the root occurrence
 4. Retrieves the active product
 5. Retrieves the rep instance
 6. Retrieves the FEM Representation and the part
 7. Retrieves the connection services object
- ...

```
Dim myConnectionService As SimConnectionServices
Set myConnectionService = MyFemRepRoot.GetItem("SimConnectionServices")
```

...

8. Get the geometrical set named : Weld Edges for CAE

...

```
Dim hander(0)
Set hander(0) = MyPart.FindObjectByName("Weld Edges for CAE")
```

...

9. Creates Line fasteners from search

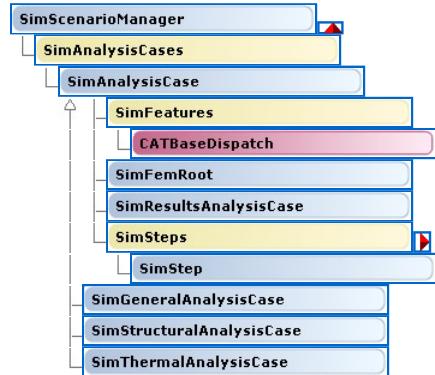
...

```
myConnectionService.CreateLinesFromSearch hander, "Test_", 10.0
```

...

Multiphysics Scenario Creation Object Model Map

See Also [Legend](#)



Use the `GetItem` method of the `SimSpecsRepManager` object to retrieve the `SimScenarioManager` object.

VBA Python

```

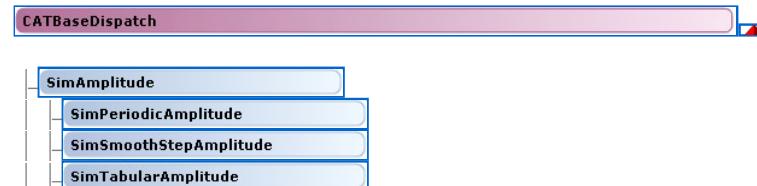
Dim oSimSpecsRepManager As SimSpecsRepManager
...
Dim oSimScenarioManager As SimScenarioManager
Set oSimScenarioManager = oSimSpecsRepManager.GetItem("SimScenarioManager")

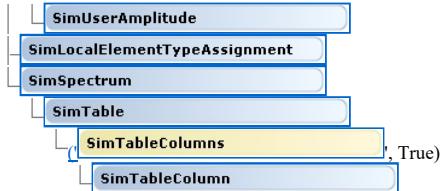
# Get the simulation editor and the simulation root
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
MySimulationRoot = myEntities.Item(1)

# Retrieve the Scenario Manager
myScenarioManager = MySimulationRoot.GetItem("SimScenarioManager")
  
```

Setup Objects

See Also [Legend](#)





Use the **Features** property of the **SimAnalysisCase** object to retrieve the **SimFeatures** collection.

VBA Python

```

Dim oSimAnalysisCase As SimAnalysisCase
...
Dim cSimFeatures As SimFeatures
Set cSimFeatures = oSimAnalysisCase.Features

...
cSimFeatures = oSimAnalysisCase.Features
...
    
```

Use the **Add** method of the **SimFeatures** collection to create a new setup feature, such as a **SimAmplitude** object.

VBA Python

```

Dim oSimAmplitude As SimAmplitude
Set oSimAmplitude = cSimFeatures.Add("SimAmplitude")

...
oSimAmplitude = cSimFeatures.Add("SimAmplitude")
...
    
```

Use the **TypeName** function to determine the type of an object retrieved from the **SimFeatures** collection in order to use its own properties and methods.

VBA Python

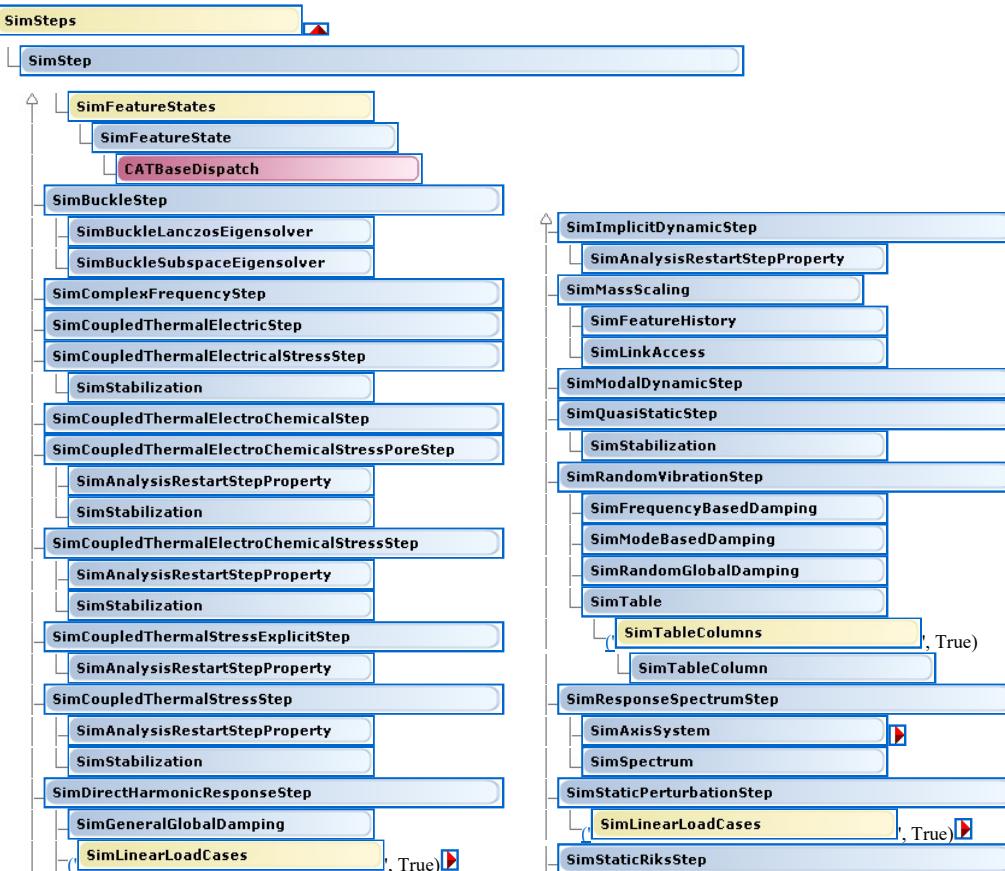
```

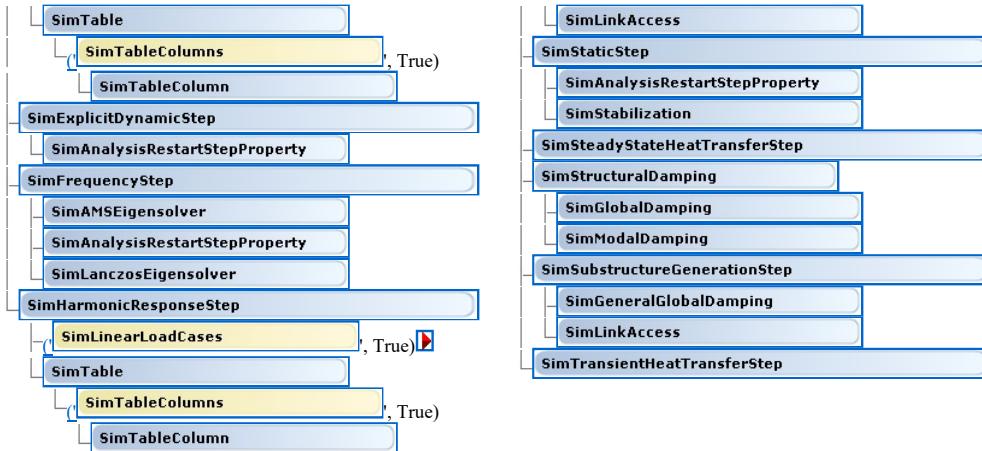
Dim oRetrievedObject As AnyObject
Set oRetrievedObject = cSimFeatures.Item(3)
Dim objectType As String
objectType = TypeName(oRetrievedObject)

...
oRetrievedObject = cSimFeatures.Item(3)
objectType = TypeName(oRetrievedObject)
...
    
```

SimSteps Collection

See Also [Legend](#)





Use the **Steps** property of the **SimAnalysisCase** object to retrieve the **SimSteps** collection.

VBA Python

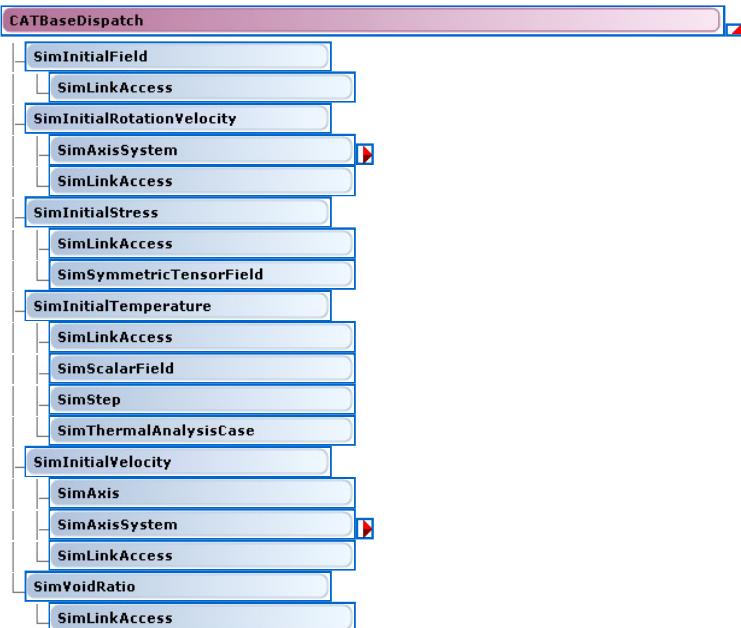
```

Dim oSimAnalysisCase As SimAnalysisCase
...
Dim cSimSteps As SimSteps
Set cSimSteps = oSimAnalysisCase.Steps

...
cSimSteps = oSimAnalysisCase.Steps
...
  
```

Initial Condition Objects

See Also [Legend](#)



Use the **Features** property of the **SimAnalysisCase** object to retrieve the **SimFeatures** collection.

VBA Python

```

Dim oSimAnalysisCase As SimAnalysisCase
...
Dim cSimFeatures As SimFeatures
Set cSimFeatures = oSimAnalysisCase.Features

...
cSimFeatures = oSimAnalysisCase.Features
...
  
```

Use the **Add** method of the **SimFeatures** collection to create a new initial condition, such as a **SimInitialStress** object.

VBA Python

```

Dim oSimInitialStress As SimInitialStress
Set oSimInitialStress = cSimFeatures.Add("SimInitialStress")

...
  
```

```
oSimInitialStress = cSimFeatures.Add("SimInitialStress")
...
```

Use the **TypeName** function to determine the type of an object retrieved from the **SimFeatures** collection in order to use its own properties and methods.

VBA Python

```
Dim oRetrievedObject As AnyObject
Set oRetrievedObject = cSimFeatures.Item(3)
Dim objectType As String
objectType = TypeName(oRetrievedObject)

...
oRetrievedObject = cSimFeatures.Item(3)
objectType = TypeName(oRetrievedObject)
...
```

Use the **GetItem** method to return the **SimLinkAccess** object from its aggregating object, such as the **SimInitialStress** object.

VBA Python

```
Dim oSimInitialStress As SimInitialStress
...
Dim oSimLinkAccess As SimLinkAccess
Set oSimLinkAccess = oSimInitialStress.GetItem("SimLinkAccess")

...
oSimLinkAccess = oSimInitialStress.GetItem("SimLinkAccess")
...
```

Interaction Objects

See Also [Legend](#)



SimGeneralContact and **SimSurfaceBasedContact** will not be documented here.

Use the **Features** property of the **SimAnalysisCase** object to retrieve the **SimFeatures** collection.

VBA Python

```
Dim oSimAnalysisCase As SimAnalysisCase
...
Dim cSimFeatures As SimFeatures
Set cSimFeatures = oSimAnalysisCase.Features

...
cSimFeatures = oSimAnalysisCase.Features
...
```

Use the **Add** method of the **SimFeatures** collection to create a new interaction, such as a **SimPortRegion** object.

VBA Python

```
Dim oSimPortRegion As SimPortRegion
Set oSimPortRegion = cSimFeatures.Add("SimPortRegion")

...
oSimPortRegion = cSimFeatures.Add("SimPortRegion")
...
```

Use the **TypeName** function to determine the type of an object retrieved from the **SimFeatures** collection in order to use its own properties and methods.

VBA Python

```
Dim oRetrievedObject As AnyObject
Set oRetrievedObject = cSimFeatures.Item(3)
Dim objectType As String
objectType = TypeName(oRetrievedObject)

...
oRetrievedObject = cSimFeatures.Item(3)
objectType = TypeName(oRetrievedObject)
...
```

Use the **GetItem** method to return the **SimLinkAccess** object from its aggregating object, such as the **SimPortRegion** object.

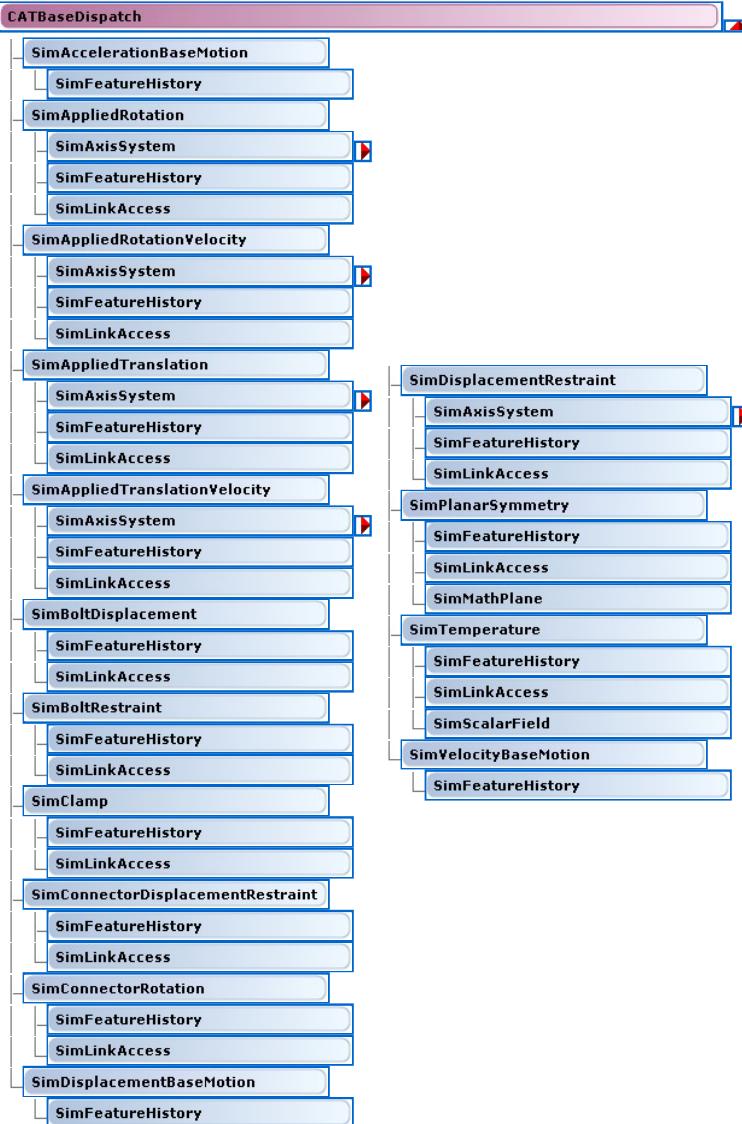
VBA Python

```
Dim oSimPortRegion As SimPortRegion
...
Dim oSimLinkAccess As SimLinkAccess
Set oSimLinkAccess = oSimPortRegion.GetItem("SimLinkAccess")

...
oSimLinkAccess = oSimPortRegion.GetItem("SimLinkAccess")
...
```

Boundary Condition Objects

See Also [Legend](#)



Use the **Features** property of the **SimAnalysisCase** object to retrieve the **SimFeatures** collection.

VBA Python

```

Dim oSimAnalysisCase As SimAnalysisCase
...
Dim cSimFeatures As SimFeatures
Set cSimFeatures = oSimAnalysisCase.Features

...
cSimFeatures = oSimAnalysisCase.Features
...
  
```

Use the **Add** method of the **SimFeatures** collection to create a new boundary condition, such as a **SimAppliedRotation** object.

VBA Python

```

Dim oSimAppliedRotation As SimAppliedRotation
Set oSimAppliedRotation = cSimFeatures.Add("SimAppliedRotation")

...
oSimAppliedRotation = cSimFeatures.Add("SimAppliedRotation")
...
  
```

Use the **TypeName** function to determine the type of an object retrieved from the **SimFeatures** collection in order to use its own properties and methods.

VBA Python

```

Dim oRetrievedObject As AnyObject
Set oRetrievedObject = cSimFeatures.Item(3)
Dim objectType As String
objectType = TypeName(oRetrievedObject)

...
oRetrievedObject = cSimFeatures.Item(3)
objectType = TypeName(oRetrievedObject)
...
  
```

Use the **GetItem** method to return the **SimLinkAccess** object from its aggregating object, such as the **SimAppliedRotation** object.

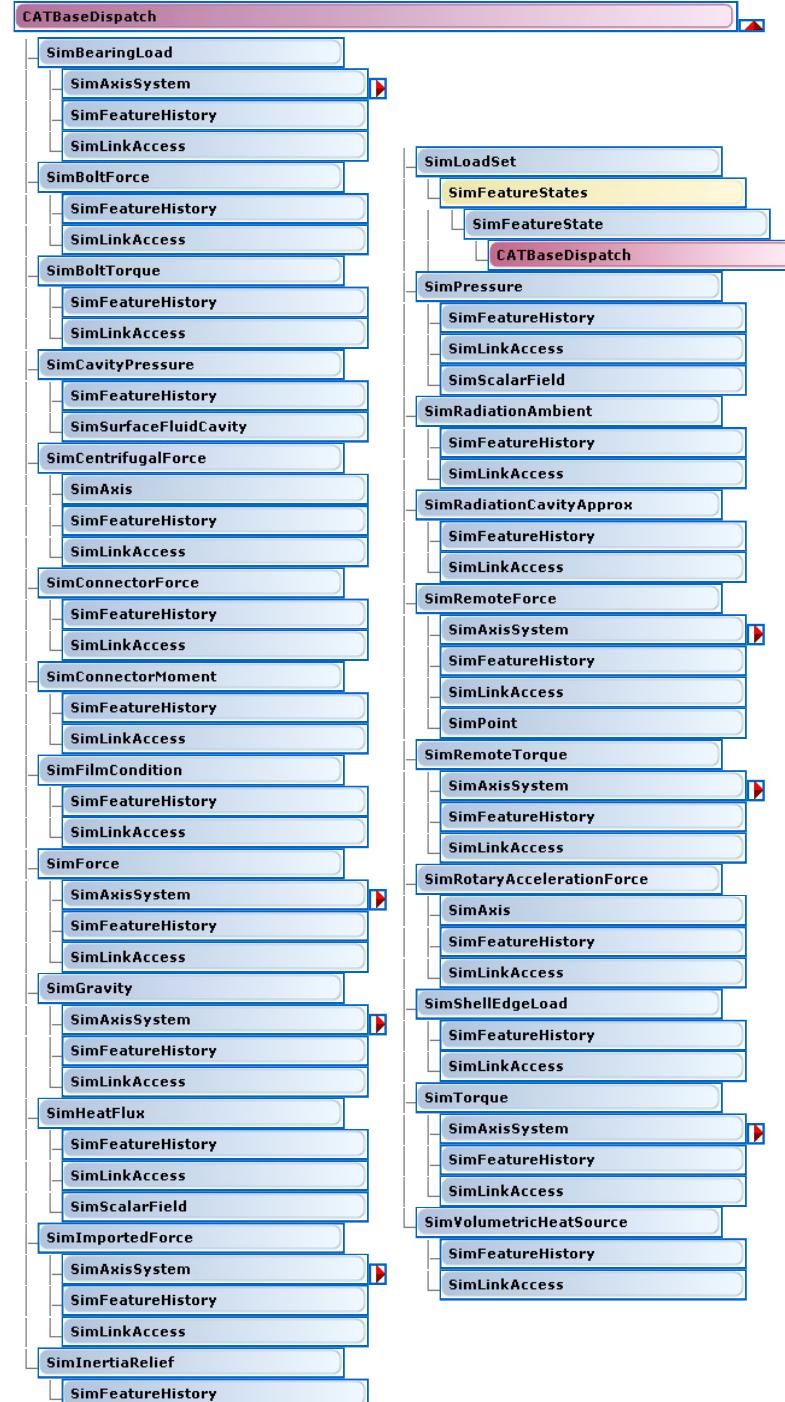
VBA Python

```
Dim oSimAppliedRotation As SimAppliedRotation
...
Dim oSimLinkAccess As SimLinkAccess
Set oSimLinkAccess = oSimAppliedRotation.GetItem("SimLinkAccess")

...
oSimLinkAccess = oSimAppliedRotation.GetItem("SimLinkAccess")
...
```

Load Objects

See Also [Legend](#)



Use the **Features** property of the **SimAnalysisCase** object to retrieve the **SimFeatures** collection.

VBA Python

```
Dim oSimAnalysisCase As SimAnalysisCase
...
Dim cSimFeatures As SimFeatures
Set cSimFeatures = oSimAnalysisCase.Features
```

```
...
cSimFeatures = oSimAnalysisCase.Features
...
```

Use the **Add** method of the **SimFeatures** collection to create a new boundary condition, such as a **SimBearingLoad** object.

VBA Python

```
Dim oSimBearingLoad As SimBearingLoad
Set oSimBearingLoad = cSimFeatures.Add("SimBearingLoad")

...
oSimBearingLoad = cSimFeatures.Add("SimBearingLoad")
...
```

Use the **TypeName** function to determine the type of an object retrieved from the **SimFeatures** collection in order to use its own properties and methods.

VBA Python

```
Dim oRetrievedObject As AnyObject
Set oRetrievedObject = cSimFeatures.Item(3)
Dim objectType As String
objectType = TypeName(oRetrievedObject)

...
oRetrievedObject = cSimFeatures.Item(3)
objectType = TypeName(oRetrievedObject)
...
```

Use the **GetItem** method to return the **SimLinkAccess** object from its aggregating object, such as the **SimBearingLoad** object.

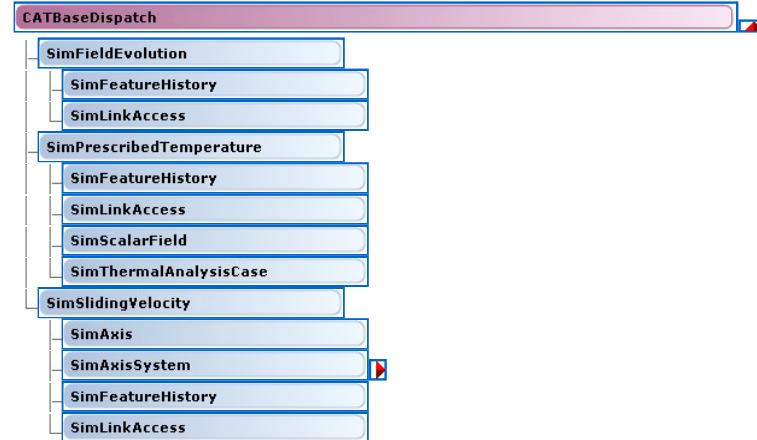
VBA Python

```
Dim oSimBearingLoad As SimBearingLoad
...
Dim oSimLinkAccess As SimLinkAccess
Set oSimLinkAccess = oSimBearingLoad.GetItem("SimLinkAccess")

...
oSimLinkAccess = oSimBearingLoad.GetItem("SimLinkAccess")
...
```

Predefined Field Objects

See Also [Legend](#)



Use the **Features** property of the **SimAnalysisCase** object to retrieve the **SimFeatures** collection.

VBA Python

```
Dim oSimAnalysisCase As SimAnalysisCase
...
Dim cSimFeatures As SimFeatures
Set cSimFeatures = oSimAnalysisCase.Features

...
cSimFeatures = oSimAnalysisCase.Features
...
```

Use the **Add** method of the **SimFeatures** collection to create a new predefined field, such as a **SimFieldEvolution** object.

VBA Python

```
Dim oSimFieldEvolution As SimFieldEvolution
Set oSimFieldEvolution = cSimFeatures.Add("SimFieldEvolution")

...
oSimFieldEvolution = cSimFeatures.Add("SimFieldEvolution")
...
```

Use the **TypeName** function to determine the type of an object retrieved from the **SimFeatures** collection in order to use its own properties and methods.

VBA Python

```

Dim oRetrievedObject As AnyObject
Set oRetrievedObject = cSimFeatures.Item(3)
Dim objectType As String
objectType = TypeName(oRetrievedObject)

...
oRetrievedObject = cSimFeatures.Item(3)
objectType = TypeName(oRetrievedObject)
...

```

Use the **GetItem** method to return the **SimLinkAccess** object from its aggregating object, such as the **SimFieldEvolution** object.

VBA Python

```

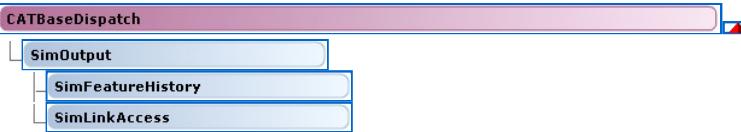
Dim oSimFieldEvolution As SimFieldEvolution
...
Dim oSimLinkAccess As SimLinkAccess
Set oSimLinkAccess = oSimFieldEvolution.GetItem("SimLinkAccess")

...
oSimLinkAccess = oSimFieldEvolution.GetItem("SimLinkAccess")
...

```

Output Objects

See Also [Legend](#)



Use the **Features** property of the **SimAnalysisCase** object to retrieve the **SimFeatures** collection.

VBA Python

```

Dim oSimAnalysisCase As SimAnalysisCase
...
Dim cSimFeatures As SimFeatures
Set cSimFeatures = oSimAnalysisCase.Features

...
cSimFeatures = oSimAnalysisCase.Features
...

```

Use the **Add** method of the **SimFeatures** collection to create a new output feature, such as a **SimOutput** object.

VBA Python

```

Dim oSimOutput As SimOutput
Set oSimOutput = cSimFeatures.Add("SimOutput")

...
oSimOutput = cSimFeatures.Add("SimOutput")
...

```

Use the **TypeName** function to determine the type of an object retrieved from the **SimFeatures** collection in order to use its own properties and methods.

VBA Python

```

Dim oRetrievedObject As AnyObject
Set oRetrievedObject = cSimFeatures.Item(3)
Dim objectType As String
objectType = TypeName(oRetrievedObject)

...
oRetrievedObject = cSimFeatures.Item(3)
objectType = TypeName(oRetrievedObject)
...

```

Use the **GetItem** method to return the **SimLinkAccess** object from its aggregating object, such as the **SimOutput** object.

VBA Python

```

Dim oSimOutput As SimOutput
...
Dim oSimLinkAccess As SimLinkAccess
Set oSimLinkAccess = oSimOutput.GetItem("SimLinkAccess")

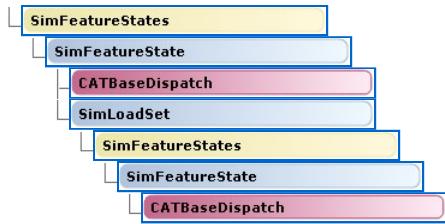
...
oSimLinkAccess = oSimOutput.GetItem("SimLinkAccess")
...

```

SimLinearLoadCases Collection

See Also [Legend](#)





Use the **LinearLoadCases** property of either the **SimHarmonicResponseStep** or **SimStaticPerturbationStep** object to retrieve the **SimLinearLoadCases** collection.

VBA Python

```

Dim oSimHarmonicResponseStep As SimHarmonicResponseStep
...
Dim cSimLinearLoadCases As SimLinearLoadCases
Set cSimLinearLoadCases = oSimHarmonicResponseStep.LinearLoadCases

...
cSimLinearLoadCases = oSimHarmonicResponseStep.LinearLoadCases
...
  
```

Creating and Executing a Simulation

This article explains how to create and execute a simulation.

Before you begin:

- Launch CATIA and then import and open the **CAAScdfeaPumpModel.3dxml** file supplied in the folder **InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples** where **InstallRootFolder** is the folder where the API CD-ROM is installed.
- Make sure that you have all of the licenses required for launching execution and perform the solve operation in the interactive session for a similar simulation to test the licenses.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaStaticSimulationSource.htm](#)

This use case can be divided in 12 steps:

1. [Retrieving the active Product](#)
2. [Retrieving the Publications and Composing Links](#)
3. [Creating the FEM Rep and Retrieving its Rep Manager](#)
4. [Creating a Mesh on a Part](#)
5. [Creating a Property](#)
6. [Initializing and Creating the Simulation PLM Object](#)
7. [Retrieving the Scenario Manager](#)
8. [Associating the Structural Analysis Case with the FEM Rep](#)
9. [Creating and Initializing the Static Step](#)
10. [Retrieving the Features Set](#)
11. [Creating features](#)
12. [Launching the Simulation](#)

1. Retrieving the active Product

As a first step, the UC retrieves the active product.

```

...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor

Dim myProdService As PLMProductService
Set myProdService = myEditor.GetService("PLMProductService")

Dim myEntities As PLMEntities
Set myEntities = myProdService.EditedContent

Dim myRootProduct As VPMReference
Set myRootProduct = myEntities.Item(1)
...
  
```

2. Retrieving the Publications and Composing Links

In this step UC retrieves the publications aggregated by the root product and composes links.

```

...
Dim myPublications As VPMPublications
Set myPublications = myRootProduct.Publications

Dim myCoverContactFace As VPMPublication
Set myCoverContactFace = myPublications.GetItem("Cover_Contact_Face")

' Composing link to the publication

Dim myRootOccurrence As VPMRootOccurrence
Set myRootOccurrence = myProdService.RootOccurrence

Dim myRepInstance As VPMRepInstance
Set myRepInstance = Nothing ' Because we use publications

Dim myLinkToCoverContactFace As AnyObject
Set myLinkToCoverContactFace = myProdService.ComposeLink(myRootOccurrence, myRepInstance, myCoverContactFace)
...
  
```

3. Creating the FEM Rep and Retrieving its Rep Manager

In this step UC creates a finite element representation model on the product which was just opened.

For further information about creating the FEM Rep and retrieving its Rep Manager, refer to the article [Creating FEM representation](#).

4. Creating a Mesh on a Part

In this step UC retrieves the nodes and elements set from the FEM Representation, creates a mesh on a part and sets the mesh attributes.

For further information about the creation and management of mesh parts, refer to the article [Creating Mesh Part](#).

5. Creating a Property

In this step UC creates a solid section on the part and sets its individual attributes.

For further information about the creation and management of properties, refer to the article [Managing FEM section features](#).

6. Initializing and Creating the Simulation PLM Object

In this step UC creates a simulation object.

```
...
' Initializing the simulation PLM Object

Dim myInitService As SimInitializationService
Set myInitService = CATIA.GetSessionService("SimInitializationService")

myInitService.SetSimulationInitialization "SimMechanicalScenarioCreation", "SimStructural"

' Creating the simulation PLM Object

Dim mySimPLMService As SIMPLMService
Set mySimPLMService = CATIA.GetSessionService("SIMPLMService")

Dim mySimulationReference As SimulationReference
mySimPLMService.PLMCreat "NewSimObject", "SMAFeaPLMNewSimu", myModel, mySimulationReference, myEditor
...
```

For further information about the initialization of a simulation object, refer to the article [Initializing a Simulation Object](#).

7. Retrieving the Scenario Manager

In this step UC retrieves the scenario manager.

```
...
Dim myScenarioManager As SimScenarioManager
Set myScenarioManager = mySimulationReference.GetItem("SimScenarioManager")
...
```

8. Associating the Structural Analysis Case with the FEM Rep

In this step UC retrieves the analysis case and associates it with the FEM rep.

```
...
Dim myAnalysisCases As SimAnalysisCases
Set myAnalysisCases = myScenarioManager.AnalysisCases

Dim myAnalysisCase As SimAnalysisCase
Set myAnalysisCase = myAnalysisCases.Item(1)

myAnalysisCase.FEMRep = myFEMRoot
...
```

9. Creating and Initializing the Static Step

In this step UC creates and initializes a static step.

```
...
Dim mySteps As SimSteps
Set mySteps = myAnalysisCase.Steps

Dim myStaticStep As SimStaticStep
Set myStaticStep = mySteps.Add("SimStaticStep")

myAnalysisCase.CreateDefaultElementTypeAssignment
myStaticStep.CreateAnalysisDefaultOutputRequests
myStaticStep.MaximumIncrements = 1000
...
```

10. Retrieving the Features Set

In this step UC retrieves the features set.

```
...
Dim myFeatures As SimFeatures
Set myFeatures = myAnalysisCase.Features

Dim myFeatureStates As SimFeatureStates
Set myFeatureStates = myStaticStep.FeatureStates
...
```

11. Creating features

In this step UC creates features for the simulation.

```
...
Dim myClamp As SimClamp
Set myClamp = myFeatures.Add("SimClamp")

' Setting feature support

Dim myLinkAccess As SimLinkAccess
Set myLinkAccess = myClamp.GetItem("SimLinkAccess")
myLinkAccess.AddLink "MainSupport", myLinkToClampSupport

' Creating feature state

myFeatureStates.CreateFeatureState myClamp
...
```

12. Launching the Simulation

In this step UC launch the simulation and shows the results.

```
...
Dim myExecutionService As SimExecutionService
Set myExecutionService = myEditor.GetService("SimExecutionService")

myExecutionService.BasicExecuteAll mySimulationReference

' Switching to the results visualization workbench

CATIA.StartWorkbench "SMAHvcResultsVisualizationWorkbench"
...
```

Initializing and Executing a Simulation

This article explains how to initialize and execute a simulation.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Launch the structural scenario App and create a structural simulation.
- Make sure that you have all of the licenses required for launching execution and perform the solve operation in the interactive session for a similar simulation to test the licenses.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaStaticSimulationWithPythonSource.htm](#)

This use case can be divided in 11 steps:

1. [Retrieves the simulation and its model](#)
2. [Retrieves the Publications and Composes Links](#)
3. [Creates the FEM Rep and retrieves its Rep Manager](#)
4. [Creates a Mesh on a Part](#)
5. [Creates a Property](#)
6. [Retrieves the Scenario Manager](#)
7. [Associates the Structural Analysis Case with the FEM rep](#)
8. [Creates and initializes the static step](#)
9. [Retrieves the features and features states sets](#)
10. [Creates features](#)
11. [Launches the simulation](#)

1. Retrieves the simulation and its model

As a first step, the UC retrieves the active simulation and its model.

```
...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...
```

2. Retrieves the Publications and Composes Links

In this step UC retrieves the publications aggregated by the root product and composes links.

```
...
myPublications = myModel.Publications
mySupport = myPublications.GetItem("Pump_Housing")

# Composing links
myLinkToSupport = myProdService.ComposeLink(None, None, mySupport)
...
```

As we use publication, we just need to give it as third argument to compose the link

3. Creates the FEM Rep and Retrieving its Rep Manager

In this step UC creates a finite element representation model on the product which was just opened.

For further information about creating the FEM Rep and retrieving its Rep Manager, refer to the article [Creating FEM representation](#).

4. Creates a Mesh on a Part

In this step UC retrieves the nodes and elements set from the FEM Representation, creates a mesh on a part and sets the mesh attributes.

For further information about the creation and management of mesh parts, refer to the article [Creating Mesh Part](#).

5. Creates a Property

In this step UC creates a solid section on the part and sets its individual attributes.

For further information about the creation and management of properties, refer to the article [Managing FEM section features](#).

6. Retrieves the Scenario Manager

In this step UC retrieves the scenario manager.

```
...
myScenarioManager = MySimulationRoot.GetItem("SimScenarioManager")
...
```

7. Associates the Structural Analysis Case with the FEM Rep

In this step UC retrieves the analysis case and associates it with the FEM rep.

```
...
myAnalysisCases = myScenarioManager.AnalysisCases
myAnalysisCase = myAnalysisCases.Item(1)
myAnalysisCase.FEMRep = myFEMRoot
...
```

8. Creates and Initializes the Static Step

In this step UC creates and initializes a static step.

```
...
mySteps = myAnalysisCase.Steps
myStaticStep = mySteps.Add("SimStaticStep")

myAnalysisCase.CreateDefaultElementTypeAssignment()
myStaticStep.CreateAnalysisDefaultOutputRequests()

myStaticStep.MaximumIncrements = 1000
...
```

9. Retrieves the Features Set

In this step UC retrieves the features set.

```
...
myFeatures = myAnalysisCase.Features
myFeatureStates = myStaticStep.FeatureStates
...
```

10. Creates features

In this step UC creates features for the simulation.

```
...
# Creates a Clamp feature
myClamp = myFeatures.Add("SimClamp")
myLinkAccess = myClamp.GetItem("SimLinkAccess")
myLinkAccess.AddLink ("MainSupport", myLinkToClampSupport)
myFeatureStates.CreateFeatureState (myClamp)
...
```

11. Launches the Simulation

In this step UC launch the simulation and shows the results.

```
...
myExecutionService = myEditor.GetService("SimExecutionService")
myExecutionService.BasicExecuteAll (MySimulationRoot)
...
```

Creating and Executing a Thermal Structural Simulation

This article explains how to create and execute a thermal structural simulation.

Before you begin: Note that:

- Launch CATIA and then import and open the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Make sure that you have all of the licenses required for launching execution and perform the solve operation in the interactive session for a similar simulation to test the licenses.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaThermalStructuralSimulationSource.htm](#)

This use case can be divided in 15 steps:

1. [Retrieving the active Product](#)
2. [Retrieving the Publications and Composing Links](#)
3. [Creating the FEM Rep and Retrieving its Rep Manager](#)
4. [Creating a Mesh on a Part](#)
5. [Creating a Property](#)
6. [Initializing and Creating the Simulation PLM Object](#)
7. [Retrieving the Scenario Manager](#)
8. [Associating the Thermal and Structural Analysis Cases with the FEM Rep](#)
9. [Creating and Initializing the Steady State Heat Transfert Step](#)
10. [Retrieving the Thermal Features Set](#)
11. [Creating Thermal features](#)
12. [Creating and Initializing the Static Step](#)
13. [Retrieving the Structural Features Set](#)
14. [Creating Structural features](#)
15. [Launching the Simulation](#)

1. Retrieving the active Product

As a first step, the UC retrieves the active product.

```
...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor

Dim myProdService As PLMProductService
Set myProdService = myEditor.GetService("PLMProductService")

Dim myEntities As PLMEntities
Set myEntities = myProdService.EditedContent

Dim myRootProduct As VPMReference
Set myRootProduct = myEntities.Item(1)
...
```

2. Retrieving the Publications and Composing Links

In this step UC retrieves the publications aggregated by the root product and composes links.

```
...
Dim myPublications As VPMPublications
Set myPublications = myRootProduct.Publications

Dim mySupport As VPMPublication
Set mySupport = myPublications.GetItem("Pump_Housing")

' Composing link to the publication

Dim myRootOccurrence As VPMRootOccurrence
Set myRootOccurrence = myProdService.RootOccurrence

Dim myRepInstance As VPMRepInstance
Set myRepInstance = Nothing ' Because we use publications

Dim myLinkToSupport As AnyObject
Set myLinkToSupport = myProdService.ComposeLink(myRootOccurrence, myRepInstance, mySupport)
...
```

3. Creating the FEM Rep and Retrieving its Rep Manager

In this step UC creates a finite element representation model on the product which was just opened.

For further information about creating the FEM Rep and retrieving its Rep Manager refer to the article [Creating FEM representation](#).

4. Creating a Mesh on a Part

In this step UC retrieves the nodes and elements set from the FEM Representation, creates a mesh on a part and sets the mesh attributes.

For further information about the creation and management of mesh parts refer to the article [Creating Mesh Part](#).

5. Creating a Property

In this step UC creates a solid section on the part and sets its individual attributes.

For further information about the creation and management of properties refer to the article [Managing FEM section features](#).

6. Initializing and Creating the Simulation PLM Object

In this step UC creates a simulation object.

```
...
' Initializing the simulation PLM Object

Dim myInitService As SimInitializationService
Set myInitService = CATIA.GetSessionService("SimInitializationService")

myInitService.SetSimulationInitialization "SimMechanicalScenarioCreation", "SimThermalStructural"

' Creating the simulation PLM Object

Dim mySimPLMService As SIMPLMService
Set mySimPLMService = CATIA.GetSessionService("SIMPLMService")

Dim mySimulationReference As SimulationReference
mySimPLMService.PLMCreate "NewSimObject", "SMAFfeaPLMNewSimu", myModel, mySimulationReference, myEditor
...
```

For further information about the initialization of a simulation object refer to the article [Initializing a Simulation Object](#).

7. Retrieving the Scenario Manager

In this step UC retrieves the scenario manager.

```
...
Dim myScenarioManager As SimScenarioManager
Set myScenarioManager = mySimulationReference.GetItem("SimScenarioManager")
...
```

8. Associating the Thermal and Structural Analysis Cases with the FEM Rep

In this step UC retrieves the analysis cases and associates it with the FEM rep.

```
...
Dim myAnalysisCases As SimAnalysisCases
Set myAnalysisCases = myScenarioManager.AnalysisCases

Dim myThermalAnalysisCase As SimAnalysisCase
Set myThermalAnalysisCase = myAnalysisCases.Item(1)

myThermalAnalysisCase.FEMRep = myFEMRoot

Dim myStructuralAnalysisCase As SimAnalysisCase
Set myStructuralAnalysisCase = myAnalysisCases.Item(2)

myStructuralAnalysisCase.FEMRep = myFEMRoot
...
```

9. Creating and Initializing the Steady State Heat Transfer Step

In this step UC creates and initializes a steady state heat transfer step.

```
...
Dim myThermalSteps As SimSteps
Set myThermalSteps = myThermalAnalysisCase.Steps

Dim mySteadyStateHeatTransferStep As SimSteadyStateHeatTransferStep
Set mySteadyStateHeatTransferStep = myThermalSteps.Add("SimSteadyStateHeatTransferStep")

myThermalAnalysisCase.CreateDefaultElementTypeAssignment

mySteadyStateHeatTransferStep.CreateAnalysisDefaultOutputRequests
mySteadyStateHeatTransferStep.TotalTime = 1.0
mySteadyStateHeatTransferStep.TimeIncrementationScheme = SimTimeIncrementation_Automatic
mySteadyStateHeatTransferStep.InitialTimeStep = 0.12
mySteadyStateHeatTransferStep.MinimumTimeStep = 0.013
mySteadyStateHeatTransferStep.MaximumTimeStep = 0.98
mySteadyStateHeatTransferStep.MaximumNumberOfIncrements = 60
...
```

10. Retrieving the Thermal Features Set

In this step UC retrieves the thermal features set.

```
...
Dim myThermalFeatures As SimFeatures
Set myThermalFeatures = myThermalAnalysisCase.Features

Dim myThermalFeatureStates As SimFeatureStates
Set myThermalFeatureStates = mySteadyStateHeatTransferStep.FeatureStates
...
```

11. Creating Thermal features

In this step UC creates thermal features for the simulation.

```
...
Dim myTemperature1 As SimTemperature
Set myTemperature1 = myThermalFeatures.Add("SimTemperature")

myTemperature1.UniformMagnitude = 320.0

' Setting feature support

Dim myLinkAccess As SimLinkAccess
Set myLinkAccess = myTemperature1.GetItem("SimLinkAccess")
myLinkAccess.AddLink "MainSupport", myLinkToTemperatureSupport1

' Creating feature state

myThermalFeatureStates.CreateFeatureState myTemperature1
...
```

12. Creating and Initializing the Static Step

In this step UC creates and initializes a static step.

```
...
Dim myStructuralSteps As SimSteps
Set myStructuralSteps = myStructuralAnalysisCase.Steps

Dim myStaticStep As SimStaticStep
Set myStaticStep = myStructuralSteps.Add("SimStaticStep")
```

```

myStructuralAnalysisCase.CreateDefaultElementTypeAssignment
myStaticStep.CreateAnalysisDefaultOutputRequests
myStaticStep.MaximumIncrements = 1000
...

```

13. Retrieving the Structural Features Set

In this step UC retrieves the thermal features set.

```

...
Dim myStructuralFeatures As SimFeatures
Set myStructuralFeatures = myStructuralAnalysisCase.Features

Dim myStructuralFeatureStates As SimFeatureStates
Set myStructuralFeatureStates = myStaticStep.FeatureStates
...

```

14. Creating Structural features

In this step UC creates features for the simulation.

```

...
Dim myClamp As SimClamp
Set myClamp = myStructuralFeatures.Add("SimClamp")

' Setting feature support

Dim myLinkAccess As SimLinkAccess
Set myLinkAccess = myClamp.GetItem("SimLinkAccess")
myLinkAccess.AddLink "MainSupport", myLinkToClampSupport

' Creating feature state

myStructuralFeatureStates.CreateFeatureState myClamp
...

```

15. Launching the Simulation

In this step UC launch the simulation and shows the results.

```

...
Dim myExecutionService As SimExecutionService
Set myExecutionService = myEditor.GetService("SimExecutionService")

myExecutionService.BasicExecuteAll mySimulationReference

' Switching to the results visualization workbench

CATIA.StartWorkbench "SMAHvcResultsVisualizationWorkbench"
...

```

Initializing and Executing a Thermal Structural Simulation

This article explains how to initialize and execute a thermal structural simulation.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Launch the structural scenario App and create a Thermal-structural simulation.
- Make sure that you have all of the licenses required for launching execution and perform the solve operation in the interactive session for a similar simulation to test the licenses.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaThermalStructuralSimulationWithPythonSource.htm](#)

This use case can be divided in 14 steps:

1. [Retrieves the simulation and its product](#)
2. [Retrieves the Publications and Composes Links](#)
3. [Creates the FEM Rep and retrieves its Rep Manager](#)
4. [Creates a Mesh on a Part](#)
5. [Creates a Property](#)
6. [Retrieves the Scenario Manager](#)
7. [Associates the Structural Analysis Case with the FEM rep](#)
8. [Creates and initializes the static step](#)
9. [Retrieves the thermal features and features states sets](#)
10. [Creates thermal features](#)
11. [Creates and initializes the static step](#)
12. [Retrieves the structural features and features states sets](#)
13. [Creates features](#)
14. [Launches the simulation](#)

1. Retrieves the simulation and its model

As a first step, the UC retrieves the active simulation and its model.

```

...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent

```

```
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...
```

2. Retrieves the Publications and Composes Links

In this step UC retrieves the publications aggregated by the root product and composes links.

```
...
myPublications = myModel.Publications
mySupport = myPublications.GetItem("Pump_Housing")

# Composing links
myLinkToSupport = myProdService.ComposeLink(None, None, mySupport)
...
```

As we use publication, we just need to give it as third argument to compose the link

3. Creates the FEM Rep and Retrieving its Rep Manager

In this step UC creates a finite element representation model on the product which was just opened.

For further information about creating the FEM Rep and retrieving its Rep Manager, refer to the article [Creating FEM representation](#).

4. Creates a Mesh on a Part

In this step UC retrieves the nodes and elements set from the FEM Representation, creates a mesh on a part and sets the mesh attributes.

For further information about the creation and management of mesh parts, refer to the article [Creating Mesh Part](#).

5. Creates a Property

In this step UC creates a solid section on the part and sets its individual attributes.

For further information about the creation and management of properties, refer to the article [Managing FEM section features](#).

6. Retrieves the Scenario Manager

In this step UC retrieves the scenario manager.

```
...
myScenarioManager = MySimulationRoot.GetItem("SimScenarioManager")
...
```

7. Associates the Structural Analysis Case with the FEM Rep

In this step UC retrieves the analysis cases and associates them with the FEM rep.

```
...
myAnalysisCases = myScenarioManager.AnalysisCases
myThermalAnalysisCase = myAnalysisCases.Item(1)
myThermalAnalysisCase.FEMRep = myFEMRoot

myStructuralAnalysisCase = myAnalysisCases.Item(2)
myStructuralAnalysisCase.FEMRep = myFEMRoot
...
```

8. Creates and initializes the steady state heat transfer step

In this step UC creates and initializes a steady state heat transfer step.

```
...
myThermalSteps = myThermalAnalysisCase.Steps

mySteadyStateHeatTransferStep = myThermalSteps.Add("SimSteadyStateHeatTransferStep")

myThermalAnalysisCase.CreateDefaultElementTypeAssignment()
mySteadyStateHeatTransferStep.CreateAnalysisDefaultOutputRequests()

mySteadyStateHeatTransferStep.TotalTime = 1.0
mySteadyStateHeatTransferStep.TimeIncrementationScheme = win32com.client.constants.SimTimeIncrementation_Automatic
mySteadyStateHeatTransferStep.InitialTimeStep = 0.12
mySteadyStateHeatTransferStep.MinimumTimeStep = 0.013
mySteadyStateHeatTransferStep.MaximumTimeStep = 0.98
mySteadyStateHeatTransferStep.MaximumNumberOfIncrements = 60
...
```

9. Retrieves the thermal features and features states sets

In this step UC retrieves the thermal features set.

```
...
myThermalFeatures = myThermalAnalysisCase.Features
myThermalFeatureStates = mySteadyStateHeatTransferStep.FeatureStates
...
```

10. Creates thermal features

In this step UC creates thermal features for the simulation.

```
...
myTemperature1 = myThermalFeatures.Add("SimTemperature")
myLinkAccess = myTemperature1.GetItem("SimLinkAccess")
myLinkAccess.AddLink ("MainSupport", myLinkToTemperatureSupport1)
```

```

myTemperature1.UniformMagnitude = 320.0
myThermalFeatureStates.CreateFeatureState (myTemperature1)
...

```

11. Creates and initializes the static step

In this step UC creates and initializes a static step.

```

...
myStructuralSteps = myStructuralAnalysisCase.Steps
myStaticStep = myStructuralSteps.Add("SimStaticStep")

myStructuralAnalysisCase.CreateDefaultElementTypeAssignment()
myStaticStep.CreateAnalysisDefaultOutputRequests()

myStaticStep.MaximumIncrements = 1000
...

```

12. Retrieves the structural features and features states sets

In this step UC retrieves the thermal features set.

```

...
myStructuralFeatures = myStructuralAnalysisCase.Features
myStructuralFeatureStates = myStaticStep.FeatureStates
...

```

13. Creates features

In this step UC creates features for the simulation.

```

...
# Creates a Clamp feature
myClamp = myStructuralFeatures.Add("SimClamp")

# Set the feature support
myLinkAccess = myClamp.GetItem("SimLinkAccess")
myLinkAccess.AddLink ("MainSupport", myLinkToClampSupport)

# Create feature state
myStructuralFeatureStates.CreateFeatureState (myClamp)
...

```

14. Launches the simulation

In this step UC launch the simulation and shows the results.

```

...
myExecutionService = myEditor.GetService("SimExecutionService")
myExecutionService.BasicExecuteAll (MySimulationRoot)

# Switching to the results visualization workbench
CATIA.StartWorkbench ("SMAHvcResultsVisualizationWorkbench")
...

```

Creating a Contact Feature

This article explains how to create a contact feature.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaContactsSource.htm](#)

This use case can be divided into six steps:

1. [Retrieving the active Product](#)
2. [Initializing and Creating the Simulation PLM Object](#)
3. [Retrieving the Features Set](#)
4. [Creating and Initializing a Contact property feature](#)
5. [Creating a Surface-based Contact feature](#)
6. [Creating a General Contact feature](#)

1. Retrieving the active Product

As a first step, the UC retrieves the active product.

```

...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor

Dim myProdService As PLMProductService
Set myProdService = myEditor.GetService("PLMProductService")

Dim myEntities As PLMEntities
Set myEntities = myProdService.EditedContent

Dim myRootProduct As VPMReference
Set myRootProduct = myEntities.Item(1)
...

```

2. Initializing and Creating the Simulation PLM Object

In this step UC initializes and creates a simulation PLM object with a static step.

For further information about initializing and creating a simulation PLM object with a static step, refer to the article [Creating and Executing a Simulation](#).

3. Retrieving the Features Set

In this step UC retrieves the features set.

```
...
Dim myFeatures As SimFeatures
Set myFeatures = myAnalysisCase.Features

Dim myFeatureStates As SimFeatureStates
Set myFeatureStates = myStaticStep.FeatureStates
...
```

4. Creating and Initializing a Contact property feature

In this step UC creates and initializes a Contact property feature.

```
...
' Creating a contact property
'-----
Dim mySurfaceBasedContactProperty As SimSurfaceBasedContactProperty
Set mySurfaceBasedContactProperty = myFeatures.Add("SimSurfaceBasedContactProperty")

' Adding a tangential behavior
'-----
mySurfaceBasedContactProperty.TangentialBehaviorFlag = TRUE

Dim myTangentialBehavior As SimTangentialBehavior
Set myTangentialBehavior = mySurfaceBasedContactProperty.TangentialBehavior

myTangentialBehavior.FrictionType = SimTangentialBehaviorCoulomb

Dim myFrictionColumn As SimTableColumn
Set myFrictionColumn = myTangentialBehavior.GetTableColumn(SimTangentialBehaviorFriction)

myFrictionColumn.SetCellData 1, 0.33
myFrictionColumn.SetCellData 2, 0.66

myTangentialBehavior.ContactPressureDependentFlag = TRUE

Dim myContactPressureColumn As SimTableColumn
Set myContactPressureColumn = myTangentialBehavior.GetTableColumn(SimTangentialBehaviorContactPressure)

myContactPressureColumn.SetCellData 1, 3
myContactPressureColumn.SetCellData 2, 4

' Adding a normal behavior
'-----
mySurfaceBasedContactProperty.NormalBehaviorFlag = TRUE

Dim myNormalBehavior As SimNormalBehavior
Set myNormalBehavior = mySurfaceBasedContactProperty.NormalBehavior

myNormalBehavior.PressureOverClosureType = SimNormalBehaviorHard
...
```

5. Creating a Surface-based Contact feature

In this step UC creates and initializes a Surface-based Contact feature.

```
...
' Creating a Surface-based contact feature
'-----
Dim mySurfaceBasedContact As SimSurfaceBasedContact
Set mySurfaceBasedContact = myFeatures.Add("SimSurfaceBasedContact")

myFeatureStates.CreateFeatureState mySurfaceBasedContact

' Setting attributes
'-----
Dim myMainSurface As SimLinkAccess
Set myMainSurface = mySurfaceBasedContact.MainSurface
myMainSurface.AddLink "MainSupport", myLinkToContactSurface1

Dim mySecondarySurface As SimLinkAccess
Set mySecondarySurface = mySurfaceBasedContact.SecondarySurface
mySecondarySurface.AddLink "MainSupport", myLinkToContactSurface2

mySurfaceBasedContact.ContactProperty = mySurfaceBasedContactProperty
...
```

6. Creating a General Contact feature

In this step UC creates and initializes a General contact feature.

```
...
' Creating a General contact feature
'-----
Dim mySimGeneralContact As SimGeneralContact
Set mySimGeneralContact = myFeatures.Add("SimGeneralContact")
```

```

myFeatureStates.CreateFeatureState mySimGeneralContact
' Setting attributes
'-----
mySimGeneralContact.SpecifyExcludedSurfacesFlag = TRUE

Dim mySurfaceExclusions As SimSurfaceExclusions
Set mySurfaceExclusions = mySimGeneralContact.SurfaceExclusions

Dim mySurfaceExclusion As SimSurfaceToSurfacePair
Set mySurfaceExclusion = mySurfaceExclusions.Add()

Dim myFirstSurface As SimLinkAccess
Set myFirstSurface = mySurfaceExclusion.FirstSurface
myFirstSurface.AddLink "MainSupport", myLinkToContactSurface1

Dim mySecondSurface As SimLinkAccess
Set mySecondSurface = mySurfaceExclusion.SecondSurface
mySecondSurface.AddLink "MainSupport", myLinkToContactSurface2
...

```

Creating a Contact Feature

This article explains how to create a contact feature.

Before you begin:

- Launch CATIA and then import and open the CAAScdFeaPumpModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Launch the structural scenario App and create a structural simulation.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdFeaContactsWithPythonSource.htm](#)

This use case can be divided into six steps:

1. [Retrieves the simulation and its product](#)
2. [Initializes the Simulation PLM Object](#)
3. [Retrieves the Features Set](#)
4. [Creates and Initializes a Contact property feature](#)
5. [Creates a Surface-based Contact feature](#)
6. [Creates a General Contact feature](#)

1. Retrieves the simulation and its product

As a first step, the UC retrieves the active product.

```

...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
MySimulationRoot = myEntity
myModel = MySimulationRoot.Model
...

```

2. Initializes the Simulation PLM Object

In this step UC initializes and creates a simulation PLM object with a static step.

For further information about initializing and creating a simulation PLM object with a static step, refer to the article [Creating and Executing a Simulation](#).

3. Retrieves the Features Set

In this step UC retrieves the features set.

```

...
myFeatures = myAnalysisCase.Features
myFeatureStates = myStaticStep.FeatureStates
...

```

4. Creates and Initializes a Contact property feature

In this step UC creates and initializes a Contact property feature.

```

...
# Creating a contact property
# -----
mySurfaceBasedContactProperty = myFeatures.Add("SimSurfaceBasedContactProperty")

# Adding a tangential behavior
# -----
mySurfaceBasedContactProperty.TangentialBehaviorFlag = True
myTangentialBehavior = mySurfaceBasedContactProperty.TangentialBehavior
myTangentialBehavior.FrictionType = win32com.client.constants.SimTangentialBehaviorCoulomb

myFrictionColumn = myTangentialBehavior.GetTableColumn(win32com.client.constants.SimTangentialBehaviorFriction)

myFrictionColumn.SetCellData (1, 0.33)
myFrictionColumn.SetCellData (2, 0.66)
myTangentialBehavior.ContactPressureDependentFlag = True
myContactPressureColumn = myTangentialBehavior.GetTableColumn(win32com.client.constants.SimTangentialBehaviorContactPressure)
myContactPressureColumn.SetCellData (1, 3)
myContactPressureColumn.SetCellData (2, 4)

```

```

# Adding a normal behavior
# -----
mySurfaceBasedContactProperty.NormalBehaviorFlag = True

myNormalBehavior = mySurfaceBasedContactProperty.NormalBehavior

myNormalBehavior.PressureOverclosureType = win32com.client.constants.SimNormalBehaviorHard
...

```

5. Creates a Surface-based Contact feature

In this step UC creates and initializes a Surface-based Contact feature.

```

...
# Creating a Surface-based contact feature
# -----
mySurfaceBasedContact = myFeatures.Add("SimSurfaceBasedContact")
myFeatureStates.CreateFeatureState (mySurfaceBasedContact)

# Setting attributes
# -----
myMainSurface = mySurfaceBasedContact.MainSurface
myMainSurface.AddLink ("MainSupport", myLinkToContactSurface1)

mySecondarySurface = mySurfaceBasedContact.SecondarySurface
mySecondarySurface.AddLink ("MainSupport", myLinkToContactSurface2)
mySurfaceBasedContact.ContactProperty = mySurfaceBasedContactProperty
...

```

6. Creates a General Contact feature

In this step UC creates and initializes a General contact feature.

```

...
# Creating a General contact feature
# -----
mySimGeneralContact = myFeatures.Add("SimGeneralContact")
myFeatureStates.CreateFeatureState (mySimGeneralContact)

# Setting attributes
# -----
mySimGeneralContact.SpecifyExcludedSurfacesFlag = True

mySurfaceExclusions = mySimGeneralContact.SurfaceExclusions
mySurfaceExclusion = mySurfaceExclusions.Add()

myFirstSurface = mySurfaceExclusion.FirstSurface
myFirstSurface.AddLink ("MainSupport", myLinkToContactSurface1)

mySecondSurface = mySurfaceExclusion.SecondSurface
mySecondSurface.AddLink ("MainSupport", myLinkToContactSurface2)
...

```

Initializing a Simulation Object

Technical Article

Abstract

The following reference section documents the required licensing for initializing various types of simulations.

- [Initializing a Simulation Object](#)

Initializing a Simulation Object

The first step in creating a new simulation is to call the `SetSimulationInitialization` method of `SimInitializationService`.

```

...
' Initializing the simulation PLM Object

Dim myInitService As SimInitializationService
Set myInitService = CATIA.GetSessionService("SimInitializationService")

myInitService.SetSimulationInitialization "SimMechanicalScenarioCreation", "SimStructural"
...

```

The argument strings into the `SetSimulationInitialization` method maps to the App and Analysis Case of the Simulation as per the table below.

App	AppName Argument	Analysis Case	TypeName Argument	Recommended Roles
SIMULIA Mechanical Scenario Creation	SimMechanicalScenarioCreation	General	SimGeneral	SYE - Structural Analysis Engineer
		Structural	SimStructural	SSU - Structural Mechanics Engineer
		Thermal	SimThermal	
		Thermal Structural	SimThermalStructural	
SIMULIA Structural Scenario Creation	SimStructuralScenarioCreation	Structural	SimStructural	SFO - Structural Performance Engineer
		Thermal	SimThermal	
		Thermal Structural	SimThermalStructural	
SIMULIA Linear Structural Scenario Creation	SimLinearStructuralScenarioCreation	Structural	SimStructural	SLL - Structural Engineer
SIMULIA Linear Structural Validation	SimLinearStructuralValidation	Structural	SimStructural	SRD - Structural Designer
		Thermal	SimThermal	
		Frequency	SimFrequency	

Buckle SimBuckle

Once the SetSimulationInitialization has been called, a new simulation can be created by calling `PLMCreate` method of **SIMPLMService**.

```
...  
' Creating the simulation PLM Object  
  
Dim mySimPLMService As SIMPLMService  
Set mySimPLMService = CATIA.GetSessionService("SIMPLMService")  
  
Dim mySimulationReference As SimulationReference  
mySimPLMService.PLMCreate "NewSimObject", "SMAFeaPLMNewSimu", myModel, mySimulationReference, myEditor  
...
```

History

Version: 1 [Dec 2015] Document created

Version: 2 [Apr 2022] Document updated to remove deprecated method

Custom Method Overview

Technical Article

Abstract

A custom Method defines a standard process for setting up a simulation for a specific purpose, making the activity easier for non-simulation-centric engineers. It facilitates a clutter-free workflow by providing only the commands that are relevant to the simulation being set up.

Capability Summary

A custom Method configures the Actions, commands, and the Assistant content displayed in the Simulation Status panel. (This is the same Simulation Status panel that is seen in the Simulation for Designers apps.) The Actions in the Simulation Status panel are selectable. Upon the selection of an Action, the Assistant content of the panel updates to show the associated help text (if any) and commands. This makes the Simulation Status panel integral to a custom Method.

Custom Methods incorporate the functionality of the Physics Simulation Scenario Creation apps. All of the commands available within the Scenario Creation apps can be used within a custom Method. (There are limitations on how the commands can be mixed together.)

The Physics Methods Reuse app does not license any simulation functionality. Therefore, for the simulation commands to be available within a custom Method, the corresponding Scenario Creation app must be licensed to the user.

Where Used

Users access the available custom Methods via the Physics Methods Reuse app.

Custom Method Definition

A custom Method is defined via an XML file. The Method file is created/edited via any xml/text editor. There is not a 3DEXPERIENCE app for creating/editing a custom Method.

Deploying Custom Methods

Deploying custom Methods and configuring the Methods that are available to users is covered in the "Physics Methods Reuse - Administration" section of the Installation and Administration guide.

History

Version: 1 [Jan 2016] Document created

Schema Documentation Conventions

Technical Article

In the custom Methods XML schema descriptions, the descriptions of the XML elements contain a "Summary" section and a "Details" section.

The Summary section uses regular expression notations:

- ? The question mark indicates there is zero or one of the attached item.
- * The asterisk indicates there is zero or more of the attached item.
- + The plus sign indicates there is one or more of the attached item.

The Details section incorporates a few constructs of standard XML schema notation (use [required, optional], minOccurs, maxOccurs, type, etc.).

History

Version: 1 [Jan 2016] Document created

Method Index XML Schema

Technical Article

Abstract

This article details the schema of the custom Method index file.

The custom Method index file is used in deploying custom Methods. Its use is covered in the "Physics Methods Reuse - Administration" section of the Installation and Administration guide (Installation and Administration | Content and Simulation | Physics Methods Reuse - Administration).

[Example index file.](#)**Summary**

```
<Index ... >
  <Method ... />+
</Index>

<Index name? nlsFile? >
  <Method name? description? nlsFile? file />
```

Details

<Index	The document root element. Required. type=xs:string Specifies the Method-group name that is displayed in the Method selection dialog. name The value is first treated as an NLS lookup key. (See nlsFile.) If the NLS lookup fails, the value is used as given. Therefore, it is not necessary to specify an NLS key. Optional - defaults to the name of the index file. (The default value is also used as a NLS key.) type=xs:string The name/path of the CATNls file for NLS lookups. (Extension omitted.) May include a relative sub-directory path. The location of the CATNls file is relative to the following: nlsFile
	<ul style="list-style-type: none"> • The directory of the index file. • The standard "resources/msgcatalog" runtime directory. <p>Optional - defaults to the basename of the index file. A CATNls file is not necessary if NLS lookups are not needed.</p>
<Method	A declaration of a custom Method. minOccurs=1 maxOccurs=unbounded type=xs:string Specifies the Method name that is displayed in the Method selection dialog. name The value is first treated as an NLS lookup key. (See nlsFile.) If the NLS lookup fails, the value is used as given. Therefore, it is not necessary to specify an NLS key. Optional - defaults to the basename of the Method file. (The default value is also used as a NLS key.) type=xs:string Specifies a description for the Method that is displayed in the Method selection dialog. description The value is first treated as an NLS lookup key. (See nlsFile.) If the NLS lookup fails, the value is used as given. Therefore, it is not necessary to specify a NLS key. Optional - defaults to "<basename of the Method file>.Description". (The default value is used only as an NLS key. If the NLS lookup fails, the description field is left empty.) type=xs:string The name/path of the CATNls file for NLS lookups. (Extension omitted.) May include a relative sub-directory path. The location of the CATNls file is relative to the following: nlsFile
	<ul style="list-style-type: none"> • The directory of the Method file. • The standard "resources/msgcatalog" runtime directory. <p>Optional - defaults to the basename of the Method file. Additionally the CATNls file associated with the <Index> element is also used. Therefore, the Method level CATNls entries can be put in the <Index> level CATNls file. A CATNls file is not necessary if NLS lookups are not needed.</p>
file	type=xs:string The path of the corresponding custom Method file. Can be either an absolute or a relative path. A relative path is relative to the path of the index file. Required.

History

Version: 1 [Jan 2016] Document created

Version: 1.1 [Feb 2016] Added link to example index file.

Custom Method XML Schema

Technical Article

Abstract

This article details the schema of the custom Method file.

A custom Method is defined via an XML file. The Method file is created/edited via any xml/text editor. There is not a **3DEXPERIENCE** app for creating/editing a custom Method.

The custom Method file uses the extension ".xml".

An actual custom Method xsd schema is located in the **3DEXPERIENCE** runtime view at

reffiles\SimulationProducts\SMAMpaMethodSchema.xsd

The xsd is not a fully validating schema. It is possible for a custom Method to contain errors that are caught only at runtime. It is strongly encouraged that you validate your custom Method instance document against this schema via a 3rd party tool.

Target Namespace

The custom Method schema is assigned the following target namespace:

urn:com.dassault_systemes.Simulia.2014.SimulationMethod

Because of the targetNamespace declaration, the custom Method instance documents must include a corresponding namespace spec. For example:

```
<Method
  xmlns="urn:com.dassault_systemes.Simulia.2014.SimulationMethod"
  ...
>
```

This namespace spec is required. A custom Method that does not contain this namespace spec is not valid.

Limitations:

1. The namespace spec in the custom Method instance documents must be specified as the default namespace. That is, it cannot specify a namespace prefix. Therefore, the namespace spec must exactly match
 xmlns="urn:com.dassault_systemes.Simulia.2014.SimulationMethod"
2. The custom Method instance documents are not validated against the schema. The consequences of this are:
 - o Extraneous XML content is not an error, but rather is quietly ignored.
 - This means that misspellings are not caught.

It is strongly encouraged that you validate your custom Method instance document via a 3rd party tool.

Method Element

Summary

```
<Method ... >
  <Assistant>? ... </Assistant>
  <EnableAcoustics>? ... </EnableAcoustics>
  <Action>+ ... </Action>
</Method >

<Method schemaVersion? type name? nlsFile? >
```

Details

<Method	The document root element. type=xs:decimal
schemaVersion	Specifies the custom Method schema version. This comes from the version attribute of the <schema> element of schema document (SMAMpaMethodSchema.xsd). Optional - defaults to 1.0.
type	Enum:[Structural, Thermal] See details below. Required.
name	type=xs:string Specifies the name that is shown in the UI. (Displayed at the top of the Action-Assistant panel). The value is first treated as an NLS lookup key. (See nlsFile.) If the NLS lookup fails, the value is used as given. Therefore, it is not necessary to specify an NLS key. Optional - defaults to the basename of the Method file. (The default value is also used as a NLS key.)
nlsFile	type=xs:string The name/path of the CATNls file for NLS lookups. (Extension omitted.) May include a relative sub-directory path. The location of the CATNls file is relative to the following: <ul style="list-style-type: none"> • The directory of the Method file. • The standard "resources/msgcatalog" runtime directory. Optional - defaults to the basename of the Method file. A CATNls file is not necessary if NLS lookups are not needed.
<Assistant>	Specifies the Method level content of the Active Assistant. See "Assistant Element". minOccurs=0 maxOccurs=1
<EnableAcoustics>	Specifies that Acoustic option should be enabled for those commands which support acoustic capability. See "EnableAcoustics Element". Note: This capability will only activate if the user has a license to a product which normally enables the acoustic command options. If no license is available a warning will be issued and the acoustic version of the commands will not be enabled. minOccurs=0 maxOccurs=1
<Action>	Specifies the Actions for this method. See "Action Element". minOccurs=1 maxOccurs=unbounded
type:	Conceptually corresponds to the intended analysis of the associated Simulation. <ul style="list-style-type: none"> • Structural - Creates a Structural Analysis Case. • Thermal - Creates a Thermal Analysis Case.

Action Element

Conceptually, an <Action> element corresponds to a workflow step. Thus, <Action> elements are used to define a workflow for setting up a Simulation.

Summary

```
<Action ... >
  <Commands>?
    <CommandHeader ... >*
      <Description> ... </Description>
    </Commands>
    <Assistant>? ... </Assistant>
  </Action >

  <Action name? nlsFile? type? id >

  <CommandHeader headerId name? iconFile? >
```

Details

<Action

- name type=xs:string
Specifies the name that is shown in the UI. (Displayed in the Action-Assistant panel).
The value is first treated as an NLS lookup key. (See nlsFile.) If the NLS lookup fails, the value is used as given. Therefore, it is not necessary to specify an NLS key.
Optional - defaults to the specified **type** for this Action. (The default value is also used as a NLS key.) If the **type** is not specified the **name** will get set to an internal default.
- nlsFile type=xs:string
The name/path of the CATNls file for NLS lookups. (Extension omitted.) May include a relative sub-directory path.
The location of the CATNls file is relative to the following:
 - The directory of the Method file.
 - The standard "resources/msgcatalog" runtime directory.
- type type=xs:string
Optional - defaults to the Method's CATNls file.
A CATNls file is not necessary if NLS lookups are not needed.
Enum:[ManageMaterials, ManageAmplitudes, ManageFem, StructuralProcedures, ThermalProcedures, StructuralControls, InitialConditions, ThermalInitialConditions, Interactions, ThermalInteractions, StructuralRestraints, StructuralLoads, ThermalConditions, Solve, Results, MeshSetup, MeshCreate, MeshOperate, MeshGroup, ModelManage, ModelProperties, ModelAbstractions, ModelConnections]
Specified to use a pre-configured (canned) Action. See notes below.
Conditionally required, mutually exclusive with the <Commands> element.
- id type=xs:string
Provides a unique id; unique within the custom Method.
This id is not visible to the user and is not used outside the context of the custom Method.
Cannot be changed after a simulation created with the custom Method is saved.
Required.
(Note: There is a case in which an id is not required. However, it is best to always specify an id.)
Contains the commands to associate with an Action. See notes below.
- <Commands> minOccurs=0 maxOccurs=1
Conditionally required, mutually exclusive with the <Action> **type** attribute.
Specifies a command that is associated with the Action.
- <CommandHeader> All associated commands are displayed in the Simulation Status panel.
minOccurs=1 maxOccurs=unbounded
- headerId type=xs:string
The command header id of the desired command. See notes below.
Required.
- name type=xs:string
The name (title) to display in the Simulation Status panel for this command. This overrides the command's built-in name.
Optional.
- iconFile type=xs:string
The file name of the icon to display in the Simulation Status panel for this command. This overrides the command's built-in icon.
Note: the icon file must be in the 3DEXPERIENCE runtime view (under resources\graphic).
Optional.
- <Description> type=xs:string
The description to display in the Simulation Status panel for this command. This overrides the command's built-in description.
minOccurs=0 maxOccurs=1
- <Assistant> minOccurs=0 maxOccurs=1
Specifies the Action level content of the Active Assistant. See "Assistant Element".

Notes

- Either the **type** attribute or the <Commands> child must be specified.
- The <Action> **type** attribute specifies a configured Action which contains a built-in list of commands. See "Commands Associated with Configured Action Types" below. The set of commands is not configurable; the built-in set of commands is always shown for a configured Action.
- The <Commands> element is used to fully customize the available commands. With the <Commands> element, only the specified commands are provided by the Action.
- The value for the <CommandHeader> **headerId** attribute is the internal 3DEXPERIENCE command header id (or command id).

See also "Discovering Command Header Identifiers".

- The only limitation on the commands that may be specified within a custom <Action> is that the command will run in the context of the Simulation being the UI-Active object. Therefore, it should be possible to use any 3DEXPERIENCE command that works correctly in the context of a Simulation.

Assistant Element

For customizing the help text that appears in the Action-Assistant panel.

Summary

```
<Assistant ... >
  <Elem>*
    <Image ... />?
    <Text ... />?
  </Elem>
  <Expander ... >*
    <Elem>*
      <Image ... />?
      <Text ... />?
    </Elem>
  </Expander>
</Assistant>

<Assistant >

<Expander title?>
```

```
<Image file width? height? />
<Text> ... <Link url> ... </Link> ... </Text>
```

Details

<Assistant	
	Specifies an entry in the Action-Assistant panel.
<Elem>	An entry is similar to a table row. minOccurs=0 maxOccurs=unbounded
<Image	Specifies an icon image to display on the left side of this entry. Use only if an icon is desired. minOccurs=0 maxOccurs=1
file	type=xs:string The name of the image file. This is expected to be a plain file name, no path. The file must be in the 3DEXPERIENCE runtime view (under resources\graphic). Required.
width	type=xs:integer Specifies the desired display width of the icon. If not specified, a default width is used. Optional.
height	type=xs:integer Specifies the desired display height of the icon. If not specified, a default height is used. Optional.
<Text> ... </Text>	Free-form textual content. This text does not use NLS lookup. This is straight, free-form, multiline text. minOccurs=0 maxOccurs=1
<Link>	Creates an in-line hyperlink. The element's text content is linked to the specified url. minOccurs=0 maxOccurs=unbounded
url	type=xs:string The hyperlink target. Required.
<Expander>	Contains a group of entries. The group can be expanded/collapsed, as a whole, on the Action-Assistant panel. minOccurs=0 maxOccurs=unbounded
title	type=xs:string Specifies the title to display next to the expander widget. Optional

Following is an example of using the <Text> and <Link> elements together.

```
<Text>inline text <Link url="...">hyperlinked text</Link> more inline text.</Text>
```

EnableAcoustics Element

This element is currently defined to be empty. Its existence specifies that the acoustic option should be enabled for those commands which support acoustic capability. Note: This capability will only activate if the user has a license to a product which normally enables the acoustic command options. If no license is available, a warning will be issued and the acoustic options of the commands will not be enabled.

The following products support acoustic analysis command options as of the R2017x release:

- SLD - Structural Vibration Analyst : Partial Acoustic Support
- SAS - Noise & Vibration Analyst : Full Acoustic Support
- SMU - Mechanical Analyst : Full Acoustic Support

History

Version: 1 [Jan 2016] Document created

Version: 1.1 [Oct 2016] Changed examples to match XSD tag sequence (Assistant first)

Custom Method Examples

Technical Article

Abstract

This article presents xml examples for custom Methods.

Examples in this article:

[Structural Method with Configured Actions](#)

[Structural Method with Custom Actions](#)

[Thermal Method with Configured Actions](#)

Structural Method with Configured Actions

This example is a complete definition of a Method for creating a structural Simulation, where all the Actions are configured Actions.

In this example all the Actions use their built in name.

```
<?xml version="1.0" encoding="utf-8"?>
<Method type="Structural"
        name="Structural Example"
        schemaVersion="1"
```

```

    xmlns="urn:com.dassault_systemes.Simulia.2014.SimulationMethod"
  >

<Assistant>
  <Elem>
    <Text>This Method creates a structural Simulation.</Text>
  </Elem>
  <Expander title="More info" >
    <Elem>
      <Text>Follow the steps above to create the Simulation.</Text>
    </Elem>
  </Expander>
</Assistant>

<Action type="ManageFem" id="ManageFem" >
  <Assistant>
    <Elem>
      <Text>Create or assign a FEM Rep to the Simulation.</Text>
    </Elem>
  </Assistant>
</Action>

<Action type="ManageMaterials" id="ManageMaterials" >
  <Assistant>
    <Elem>
      <Text>Apply material if there are unmaterialed Parts.</Text>
    </Elem>
  </Assistant>
</Action>

<Action type="InitialConditions" id="InitialConditions" />

<Action type="StructuralProcedures" id="StructuralProcedures" >
  <Assistant>
    <Elem>
      <Text>Create the desired procedure.</Text>
    </Elem>
  </Assistant>
</Action>

<Action type="ManageAmplitudes" id="ManageAmplitudes" />

<Action type="Interactions" id="Interactions" />

<Action type="StructuralControls" id="StructuralControls" />

<Action type="StructuralRestraints" id="StructuralRestraints" >
  <Assistant>
    <Elem>
      <Text>Add restraints.</Text>
    </Elem>
  </Assistant>
</Action>

<Action type="StructuralLoads" id="StructuralLoads" >
  <Assistant>
    <Elem>
      <Text>Add loads.</Text>
    </Elem>
  </Assistant>
</Action>

<Action type="Solve" id="Solve" >
  <Assistant>
    <Elem>
      <Text>Solve the Simulation.</Text>
    </Elem>
  </Assistant>
</Action>

<Action type="Results" id="Results" >
  <Assistant>
    <Elem>
      <Text>Verify that the results are good.</Text>
    </Elem>
  </Assistant>
</Action>
</Method>

```

Structural Method with Custom Actions

This example is a complete definition of a Method for creating a structural Simulation, that contains custom Actions for the procedure, restraints, and loads. Also shows usage of the **name** attribute and the **<Description>** child of the **<CommandHeader>** element.

```

<?xml version="1.0" encoding="utf-8"?>

<Method type="Structural"
  name="Structural Example"
  schemaVersion="1"

  xmlns="urn:com.dassault_systemes.Simulia.2014.SimulationMethod"
>

<Assistant>
  <Elem>
    <Text>This Method creates a structural Simulation.</Text>
  </Elem>
  <Expander title="More info" >
    <Elem>
      <Text>Follow the steps above to create the Simulation.</Text>
    </Elem>
  </Expander>
</Assistant>

```

```
<Action type="ManageFem" id="ManageFem" />
<Action type="ManageMaterials" id="ManageMaterials" />
<Action name="Create Procedure" id="CreateProcedure" >
  <Assistant>
    <Elem>
      <Text>Create a Static Stress procedure.</Text>
    </Elem>
  </Assistant>
  <Commands>
    <CommandHeader headerId="SMAExsStaticStressHdr" />
    <CommandHeader headerId="SMAExsEditCurrentStepHdr" />
    <CommandHeader headerId="SMAExsDeleteCurrentStepHdr" />
  </Commands>
</Action>
<Action type="ManageAmplitudes" id="ManageAmplitudes" />
<Action type="InitialConditions" id="InitialConditions" />
<Action type="Interactions" id="Interactions" />
<Action type="StructuralControls" id="StructuralControls" />
<Action name="Apply Restraint" id="ApplyRestraint" >
  <Assistant>
    <Elem>
      <Text>Add restraints</Text>
    </Elem>
  </Assistant>
  <Commands>
    <CommandHeader headerId="SMAExsClampHdr" name="Clamp Face" >
      <Description>Apply a clamp to the face on the right side.</Description>
    </CommandHeader>
    <CommandHeader headerId="SMAExsDisplacementRestraintHdr" />
  </Commands>
</Action>
<Action name="Apply Load" id="ApplyLoad" >
  <Assistant>
    <Elem>
      <Text>Add loads</Text>
    </Elem>
  </Assistant>
  <Commands>
    <CommandHeader headerId="SMAExsPressureHdr" name="Side Pressure" >
      <Description>Apply a pressure to the face on the left side.</Description>
    </CommandHeader>
    <CommandHeader headerId="SMAExsRemoteForceHdr" />
  </Commands>
</Action>
<Action type="Solve" id="Solve" />
<Action type="Results" id="Results" />
</Method>
```

Thermal Method with Configured Actions

This example is a complete definition of a Method for creating a thermal Simulation, where all the Actions are configured Actions.

In this example all the Actions use their built in name.

```
<?xml version="1.0" encoding="utf-8"?>
<Method type="Thermal"
       name="Thermal Example"
       schemaVersion="1"
       xmlns="urn:com.dassault_systemes.Simulia.2014.SimulationMethod"
       >

  <Assistant>
    <Elem>
      <Text>This Method creates a thermal Simulation.</Text>
    </Elem>
    <Expander title="More info" >
      <Elem>
        <Text>Follow the steps above to create the Simulation.</Text>
      </Elem>
    </Expander>
  </Assistant>

  <Action type="ManageFem" />
  <Action type="ManageMaterials" />
  <Action type="ThermalProcedures" />
  <Action type="ManageAmplitudes" />
  <Action type="ThermalInitialConditions" />
  <Action type="ThermalInteractions" />
  <Action type="ThermalConditions" />
  <Action type="Solve" />
  <Action type="Results" />
</Method >
```

History

Version: 1 [Feb 2016] Document created
 Version: 1.1 [Oct 2016] Changed examples to match XSD tag sequence (Assistant first)

VB Scripting Integration

Technical Article

Abstract

This article details how to incorporate VB scripting into a custom Method.

Command to Run a VB Script

The SMAExsExecuteScriptCmdHdr command can be used to set up a command that executes a VB script.

The only limitation on VB script behavior is that the script will run in the context of the Simulation being the UI-Active object. Therefore, it should be possible to use any VB scripting functionality that works correctly in the context of a Simulation.

The details of the script to run are provided via an additional <ConfigSpec> schema element child of the <CommandHeader> element.

ConfigSpec Element

Provides the configuration information that identifies the script to execute.

Summary

```
<CommandHeader>
  <ConfigSpec ... />
</CommandHeader>

<ConfigSpec libraryName libraryRevision? libraryType scriptName functionName? >
```

Details

<ConfigSpec

<code>libraryName</code> type=xs:string The name of the 3DEXPERIENCE macro library containing the intended script. This is the library's PLM name (or id) (This is separate from the library's title. The title can be modified, the name cannot.) Required.	<code>libraryRevision</code> The desired revision of the PLM library. Optional - needed only if the library has multiple revisions. Enum:[PLM Directory, PLM VBA Project, PLM VSTA Project]
<code>libraryType</code> See details below. Required.	<code>scriptName</code> The name of the intended script. This is the name as displayed within the 3DEXPERIENCE Macro Library UI. Required.
<code>functionName</code> The name of the function to run within the specified script. (The script may be either a function or a subroutine; there is no distinction between the two.) Optional - defaults to "CATMain".	

The library types are:

PLM Directory	A 3DEXPERIENCE PLM macro library that contains catvbs or catscript files. The type name displayed within the 3DEXPERIENCE UI could be either "PLM Directory" or "Macro library VB Script".
PLM VBA Project	A 3DEXPERIENCE VBA PLM macro library. The type name displayed within the 3DEXPERIENCE UI could be either "PLM VBA Project" or "Macro library VBA".
PLM VSTA Project	A 3DEXPERIENCE VSTA PLM macro library. The type name displayed within the 3DEXPERIENCE UI could be either "PLM VSTA Project" or "Macro library VSTA".

Example

```
<Action name="Initialize FEM" id="InitFem" >
  <Commands>
    <CommandHeader headerId="SMAExsExecuteScriptCmdHdr" >
      <ConfigSpec libraryName="vbs 41723085-00000001"
                   libraryType="PLM Directory"
                   scriptName="InitFem1.CATScript"
                 />
    </CommandHeader>
  </Commands>
</Action>
```

Additional Scripting Functionality

The ExecuteScript command provides the following functionalities:

- User input parameters: a script can request user inputs.
- "let" subroutine: a second script subroutine can be used to
 - Get the default/initial values for the user inputs.

- Block the call of the main script.
- Manage features: a script subroutine can return the Simulation preprocessing features it creates.

Script User Input Parameters

Overview

The ExecuteScript command provides a mechanism for a script to request user inputs. In such a case, a dialog is displayed for the user to provide the requested inputs.

The values that the user enters are provided as arguments to the script subroutine.

The supported input types are:

- Simple types: integer, double, string, Boolean
- Literals: length, angle, etc.
- Enum: a locally defined fixed set of choices.
- Publication (Port).
- Meshing rules document.
- Face, Edge or Node Selection.

This functionality leverages the user inputs functionality of the Automated Modeling (batch meshing) Procedures. See Automated Modeling User Procedures for details (Content and Simulation Apps | Physics Simulation | Model Assembly Design | Automated Modeling | About User Procedures).

Limitation:

The Script User Input Parameters feature does not work with a VSTA project script. The script must be either a PLM Directory script or a VBA Project script.

If Cancel is selected on the dialog, the script execution is aborted.

Details

A script requests user inputs by adding a specific comment header, called a user input specification, to the script:

```
'<Inputs>
'  <Input Name="Input1" Type="Integer" />
'  <Input Name="Input2" Type="Double" />
'</Inputs>
```

The displayed dialog will include a field for each input requested. The following table gives the default values for the dialog fields.

Input type	Dialog default value
Integer, Double	0 (zero)
String, Dimension, DmtDocument	"" (empty string)
Boolean	False
Port	"" (empty string)
Enum	The first enum entry
Selection	An empty list

The initial value to use for each field of the dialog can be specified via the secondary 'let' subroutine of the ExecuteScript command.

The script subroutine that is being called specifies an input parameter for each input requested. That is, for the input specification header given above, the script subroutine would be defined as

```
Sub CATMain( P1, P2 )
  (any names can be used for 'P1' and 'P2'.)
```

User Input Specification Schema

Summary

```
<Enums>
  <Enum ... /**
    <Member ... /**
  </Enum>
</Enums>

<Inputs>
  <Input ... /**
</Inputs>

<Enum Name >

<Member Name Nls >

<Input Name Type SubType? Mand Nls />
```

Details

- | | |
|-----------------------------|---|
| <Enums> | Container for <Enum> specifications. |
| <Enum ... />* | Omitted if no <Enum> specifications are given. |
| | minOccurs=0 maxOccurs=1 |
| | Defines an enumeration. |
| <Input ... />* | An enumeration defines a fixed set of choices for the possible values of an input field. |
| | minOccurs=0 maxOccurs=unbounded |
| | type=xs:string |
| Name | The internal name of the enumeration. |
| | This is the name by which the enumeration is referenced in an <Input> element. |

<Member	Defines a member of an enumeration. minOccurs=0 maxOccurs=unbounded type=xs:string Name	The internal name of the member. The user does not see this name. Required.
Nls	type=xs:string The name that is displayed to the user. Required.	
<Inputs>	Container for <Input> specifications. minOccurs=1 maxOccurs=1	
<Input	Specifies a desired user input. minOccurs=0 maxOccurs=unbounded type=xs:string Name	The internal name of this input item. Required.
		enum: [Integer, Double, String, Boolean, Dimension, Port, Enum, DmtDocument] The type of the input field. Required.
		Dimension, Enum, and DmtDocument must be further qualified by the SubType attribute.
Type	Dimension represents a double value of a particular 3DEXPERIENCE dimension type. Port is for selecting a Publication. The dialog displays a list of the Publications that are defined on the Simulation's associated Product. DmtDocument is for selecting a meshing rules document. Selection is for selecting a 1 or more Nodes, Edges or Faces.	
		type=xs:string Qualifies the Dimension, Enum, and DmtDocument types. Optional required: Required if Type is Dimension, Enum, or DmtDocument.
		For Dimension, the value is a 3DEXPERIENCE dimension name. I.e., Length, Angle.
		For Enum, the value is the name of the Enum to reference. (The Enum name is given by the Name attribute of the <Enum> element.)
		For the DmtDocument, value is the extension of the meshing rules document (the file extension required for each type of meshing rule). For example: SubType CATFmtOct2dRule, CATFmtOct3dRule, CATFmtSurfTriaRule, CATFmtSurfQuadRule, CATFmtTetRule.
		For Selection, value is the type of entities to select:
		<ul style="list-style-type: none"> MonoSelectionNode - Allow selection of a single node. MonoSelectionEdge - Allow selection of a single edge. MultiSelectionEdge - Allow selection of multiple edges. MonoSelectionFace - Allow selection of a single face. MonoSelectionNode - Allow selection of a single node MultiSelectionFace - Allow selection of multiple faces.
Mand	type=xs:boolean Indicates whether the user must enter a value for this input. Optional. Defaults to false.	
Nls	type=xs:string Specifies the field name that is displayed on the dialog. Required.	

VB API

For the called subroutine, the type of each input parameter is based on the type of the corresponding input request, as given in the following table.

User Input Type	VB Parameter Type
Integer	Integer
Double	Single
String	String
Boolean	Boolean
Dimension	String, containing the numeric value and the associated current UI units (this string may not be the actual value and units that the user entered)
Enum	Integer, representing the ordinal value of the selected enum (1, 2, etc)
Port	VPMPublication (Object) * will be Empty if a Publication is not actually selected on the Inputs dialog.
DmtDocument	String, the document name. * will be an empty string if a document is not actually selected on the Inputs dialog.
Selection	Collection of AnyObject Actual object type in collection changes based on selection of Node, Edge or Face.

Example

```
'<Enums>
'  <Enum Name="Color">
'    <Member Name="red"  Nls="Red"/>
'    <Member Name="green" Nls="Green"/>
'    <Member Name="blue"  Nls="Blue"/>
'  </Enum>
'</Enums>

'<Inputs>
'  <Input Name="Input01" Type="Integer" Mand="false" Nls="Integer" />
'  <Input Name="Input02" Type="Double"  Mand="false" Nls="Double" />
'  <Input Name="Input03" Type="String"   Mand="false" Nls="String" />
'  <Input Name="Input04" Type="Boolean"  Mand="false" Nls="On/Off" />
```

```

' <Input Name="Input05" Type="Dimension" SubType="Length" Mand="false" Nls="Length" />
' <Input Name="Input06" Type="Enum" SubType="Color" Mand="false" Nls="Enum: Color" />
' <Input Name="Input07" Type="Port" Mand="false" Nls="Publication" />
' <Input Name="Input08" Type="DmtDocument" SubType="CATFmtSurfTriaRule" Mand="false" Nls="Mesh Rule" />
'</Inputs>

Sub CATMain( P1, P2, P3, P4, P5, P6, P7, P8 )

    Dim str As String
    str = ""
    str = str & "Integer = " & P1 & Chr(13)&Chr(10)
    str = str & "Double = " & P2 & Chr(13)&Chr(10)
    str = str & "String = " & P3 & Chr(13)&Chr(10)
    str = str & "Boolean = " & P4 & Chr(13)&Chr(10)
    str = str & "Length = " & P5 & Chr(13)&Chr(10)
    str = str & "Color = " & P6 & Chr(13)&Chr(10)

    Dim FeatureName As String
    FeatureName = "<None>"
    Dim P7Type As String
    P7Type = TypeName(P7)
    If Not IsEmpty(P7) Then
        If Not P7 Is Nothing Then
            FeatureName = P7.Name
        End If
    End If

    str = str & "Port = " & FeatureName & Chr(13) & Chr(10)
    str = str & "Mesh Rule = " & P8 & Chr(13)&Chr(10)

    MSGBOX str
End Sub

```

Let Subroutine

The ExecuteScript command supports a second subroutine that provides an opportunity to:

- block the call of the main script subroutine, or
- provide default/initial values for the user inputs dialog.

This subroutine is referred to as a "let" subroutine. As implied, this subroutine is called before the main subroutine (CATMain) is called.

Limitation:
The let subroutine feature does not work with scripts in a PLM Directory library.

Let Subroutine Schema

The let subroutine is specified within the ExecuteScript command's associated <ConfigSpec> element, via the following schema:

Summary

<ConfigSpec letSub? >

Details

<ConfigSpec
 type=xs:string
 letSub The name of the "let" subroutine. (Similar to the functionName attribute.)
 The let subroutine is in the same script as the main subroutine.
 Optional.

Note: The ExecuteScript command has only a single <ConfigSpec> element.

VB API

The let subroutine has the following prototype:

```
Sub LetFunction( ilFeatures(), olParams(), obBlockCall )
```

Where

param ilFeatures [in]

The list of features previously created by the main script subroutine (e.g. CATMain).

This is the list of features returned by the main subroutine on the last invocation.

If the "manageFeatures" option (described below) is not being used, this parameter will be a zero-length array.

param olParams [in/out]

List of default/initial values to use on the user inputs dialog.

Set the values as desired.

The content of the array is based on the User Inputs Specification at the top of the script module.

The values are initialized with the internal default for each input type.

For Port values, the returned value can be either an actual VMPublication object or a Publication name. In the case of a Publication name, the name will be resolved to the corresponding VMPublication object for input to the main script subroutine.

param obBlockCall [out]

Integer value passed back by this script that indicates whether to not call the main script subroutine.

0: do not block; run the main subroutine.

1: Do not run the main subroutine.

Otherwise: Do not run the main subroutine, and display an info message.

Example

Assume the following User Input Specification:

```
'<Enums>
'  <Enum Name="Color">
'    <Member Name="red"   Nls="Red"/>
'    <Member Name="green" Nls="Green"/>
'    <Member Name="blue"  Nls="Blue"/>
'  </Enum>
'</Enums>

'<Inputs>
'  <Input Name="Input01" Type="Integer" Mand="false" Nls="Integer" />
'  <Input Name="Input02" Type="Double"  Mand="false" Nls="Double" />
'  <Input Name="Input03" Type="String"   Mand="false" Nls="String" />
'  <Input Name="Input04" Type="Boolean"  Mand="false" Nls="On/Off" />
'  <Input Name="Input05" Type="Dimension" SubType="Length" Mand="false" Nls="Length" />
'  <Input Name="Input06" Type="Enum" SubType="Color" Mand="false" Nls="Enum: Color" />
'  <Input Name="Input07" Type="Port" Mand="false" Nls="Publication" />
'  <Input Name="Input08" Type="DmtDocument" SubType="CATFmtSurfTriaRule" Mand="false" Nls="Mesh Rule" />
'</Inputs>

Sub CATMain( P1, P2, P3, P4, P5, P6, P7, P8 )
  ...
End Sub
```

The let subroutine is defined as follows, in the same script files as CATMain:

```
Sub LetFunction1( ilFeatures(), olParams(), oBlockCall )

  olParams(0) = 123
  olParams(1) = 234
  olParams(2) = "This is it."
  olParams(3) = True
  olParams(4) = "456mm"
  olParams(5) = 3          'corresponds to Color Blue.
  olParams(6) = "Clamp"    'The name of the desired Publication.
  olParams(7) = "ThisOne"  'The name of the rule document.

End Sub
```

Following is the setup of the ExecuteScript command:

```
<Action name="Initialize FEM" id="InitFem" >
  <Commands>
    <CommandHeader headerId="SMAExsExecuteScriptCmdHdr" >
      <ConfigSpec libraryName="vbs 41723085-00000001"
                   libraryType="PLM Directory"
                   scriptName="InitFem1.CATscript"
                   letSub="LetFunction1"
      />
    </CommandHeader>
  </Commands>
</Action>
```

Manage Features

The ExecuteScript command supports an option to enable the script to return the Simulation preprocessing features that the script creates. This is called the "manage features" option.

This list of features is provided as input to the secondary 'let' subroutine described above.

Manage Features Schema

The "manage features" option is specified within the ExecuteScript command's associated <ConfigSpec> element, via the following schema:

Summary

<ConfigSpec manageFeatures? >

Details

<ConfigSpec
 type=xs:boolean
 manageFeatures Indicates whether the "manageFeatures" option is active for the main subroutine (e.g. CATMain).
 Optional. Defaults to false.

Note: The ExecuteScript command has only a single <ConfigSpec> element.

VB API

If the "manage features" option is specified, the main script subroutine must include an array input/output parameter. For example:

```
Sub CATMain( iolFeatures() )
```

If the "manage features" option is used along with the User Input Specification, the features array is the last parameter of the subroutine. For example:

```
Sub CATMain( P1, P2, iolFeatures() )
```

On the first call of the main subroutine, the passed in argument is a zero-length array. To return the created features, first the array size must be set, as in:

```
ReDim iolFeatures(d)
```

where "d" specifies the size of the array (see the ReDim command).

Example

Assume the following User Input Specification:

```
'<Inputs>
'  <Input Name="Input01" Type="Integer" Mand="false" Nls="Integer" />
'  <Input Name="Input02" Type="Double"  Mand="false" Nls="Double" />
'</Inputs>
```

The main subroutine is defined as follows:

```
Sub CATMain( P1, P2, iolFeatures() )
  ...
  ReDim iolFeatures(0)
  Set iolFeatures(0) = myFeature1
End Sub
```

Following is the setup of the ExecuteScript command:

```
<Action name="Initialize FEM" id="InitFem" >
  <Commands>
    <CommandHeader headerId="SMAExsExecuteScriptCmdHdr" >
      <ConfigSpec libraryName="vbs 41723085-00000001"
                    libraryType="PLM Directory"
                    scriptName="InitFem1.CATscript"
                    manageFeatures="true"
      />
    </CommandHeader>
  </Commands>
</Action>
```

History

Version: 1 [Jan 2016] Document created

Version: 1.1 [Feb 2016] Added examples

Version: 1.2 [Jul 2016] Added selection keywords to input options

Commands Associated with Configured Action Types

Technical Article

Abstract

This article details the commands associated with configured Action types.

The following table lists the commands that are provided by each of the configured Action types. These commands can be specified within custom <Action>s.

Limitation: A custom <Action> should contain commands from only a single configured Action type. Commands from different Action types should not be included together.

Exception: The commands of the Action types marked with an asterisk (*) may be included together.

A description of each command is given in addendum "Description of Simulation Commands".

	Action Type	CommandHeaders
ManageFem	SMAExsSelectFemCmdHdr SMAExsNewFemCmdHdr SMAExsSwitchToModelerHdr SMAExsElementTypeAssignmentHdr SMAFeaAssignMaterialCmdHdr SMAFeaCreateMaterialCmdHdr SMAFeaEditAppliedMatCmdHdr SMAFeaPickerMatCmdHdr SMAFeaDeleteAppliedMatCmdHdr SMAExsContactSearchHdr SMAExsContactPairHdr SMAExsGeneralContactHdr SMAExsContactPropertyHdr SMAExsContactInitializationHdr SMAExsContactInterferenceHdr SMAExsContactControlsHdr SMAExsDisconnectedPartsHdr SMAExsTouchContactHdr SMAExsIntersectContactHdr SMAExsChangeFrictionHdr SMAExsThermalContactSearchHdr SMAExsThermalContactPairHdr SMAExsThermalContactPropertyHdr	
ManageMaterials		
Interactions		
Applicability: Structural Analysis Case		
ThermalInteractions		

Applicability: Thermal Analysis Case	SMAExsTouchContactHdr SMAExsIntersectContactHdr SMAExsEditCurrentStepHdr SMAExsDeleteCurrentStepHdr SMAExsStaticStressHdr SMAExsViscoStressHdr SMAExsImplicitDynamicStressHdr SMAExsExplicitDynamicStressHdr SMAExsStaticStressPerturbationHdr SMAExsFrequencyHdr SMAExsComplexFrequencyHdr SMAExsHarmonicResponseHdr SMAExsBuckleHdr SMAExsModalDynamicHdr SMAExsEditCurrentStepHdr SMAExsDeleteCurrentStepHdr SMAExsSteadyStateHeatTransferHdr SMAExsTransientHeatTransferHdr SMAExsTabularAmplitudeHdr SMAExsSmoothStepAmplitudeHdr SMAExsPeriodicAmplitudeHdr SMAExsInitialTemperatureHdr SMAExsInitialTranslationalVelocityHdr SMAExsInitialRotationalVelocityHdr SMAExsVoidRatioHdr
StructuralProcedures	
Applicability: Structural Analysis Case	
ThermalProcedures	
Applicability: Thermal Analysis Case	
ManageAmplitudes *	
InitialConditions *	
Applicability: Structural Analysis Case	
StructuralControls *	
Applicability: ExplicitDynamicStress, HarmonicResponse, ModalDynamic Procedures	SMAExsMassScalingHdr SMAExsDampingHdr
StructuralRestraints *	SMAExsClampHdr SMAExsDisplacementRestraintHdr SMAExsPlanarSymmetryHdr SMAExsBoltRestraintHdr SMAExsBallJointHdr SMAExsSliderHdr SMAExsHingeHdr SMAExsConnectorDisplacementRestraintHdr SMAExsPressureHdr SMAExsBearingLoadHdr SMAExsCavityPressureHdr SMAExsForceHdr SMAExsImportedForceHdr SMAExsRemoteForceHdr SMAExsTorqueHdr SMAExsRemoteTorqueHdr SMAExsGravityHdr SMAExsInertiaReliefHdr SMAExsBoltLoadHdr SMAExsBoltTorqueHdr SMAExsPrescribedBoltDisplacementHdr SMAExsAppliedTranslationHdr SMAExsAppliedRotationHdr
Applicability: Structural Analysis Case	SMAExsAppliedTranslationalVelocityHdr SMAExsAppliedRotationalVelocityHdr SMAExsAppliedTranslationalAccelerationHdr SMAExsAppliedRotationalAccelerationHdr SMAExsPrescribedTemperatureHdr SMAExsLoadSetHdr SMAExsLoadCaseHdr SMAExsCentrifugalForceHdr SMAExsRotaryAccelerationLoadHdr SMAExsDisplacementBaseMotionHdr SMAExsVelocityBaseMotionHdr SMAExsAccelerationBaseMotionHdr SMAExsConnectorForceHdr SMAExsConnectorMomentHdr SMAExsConnectorRelativeDisplacementHdr SMAExsConnectorRelativeRotationHdr SMAExsSlidingVelocityHdr
StructuralLoads *	
Applicability: Structural Analysis Case	
ThermalInitialConditions *	
Applicability: Thermal Analysis Case	SMAExsInitialTemperatureHdr
ThermalConditions *	
Applicability: Thermal Analysis Case	SMAExsTemperatureHdr SMAExsHeatFluxHdr SMAExsFilmConditionHdr SMAExsVolumetricHeatSourceHdr SMAExsRadiationAmbientHdr SMAExsRadiationCavityApproxHdr SMAExsFieldOutputRequestHdr SMAExsSimulateHdr †

† As of R2016x, this command is no longer used in the Physics Simulation Scenario Creation apps. Usage of this command is not supported on cloud and is deprecated.

As of R2016x FD03 maintenance release, use the following expert Scenario Createion app commands:

SMAExsSimulationExecutionModelChecksCmdHdr
SMAExsSimulationExecutionSolverChecksCmdHdr
SMAExsSimulationExecutionFullSolveCmdHdr

SMAHvcResultsVisualizationHdr
SMAHvcCreateContourPlotHdr
SMAHvcCreateSymbolPlotHdr
SMAHvcCreateIsocontourPlotHdr
SMAHvcCreateColorCodePlotHdr
SMAHvcCreateHistoryPlotHdr
SMAHvcImportHistoryCurveHdr
SMAHvcCreateXYFromPathPlotHdr
SMAHvcCreateSensorHdr
SMAHvcCreateResultantSensorHdr
SMAHvcCreateFrequencySensorHdr
SMAHvcCreateBuckleModeSensorHdr
SMAHvcStreamHdr
SMAHvcDisplayGroupInitializeHdr
SMAHvcCreateReportHdr
Results SMAHvcCreateEnvelopeHdr
SMAHvcCreateUserCsysHdr
SMAHvcCreatePathPlotHdr
SMAHvcPlayAnimationHdr
SMAHvcCreatePlyStackPlotHdr
SMAHvcExtremaToolbarHdr
SMAHvcCutToolbarGPAHdr
SMAHvcMultiVPFourSquareHdr
SMAHvcMultiVPTriLeftHdr
SMAHvcMultiVPTriBottomHdr
SMAHvcMultiVPTwoRowHdr
SMAHvcMultiVPTwoColumnHdr
SMAHvcMultiVPOffHdr
SMAHvcMultiVPSettingsHdr
SMAHvcGlobalOptionsHdr
SMAHvcRenderingSettingsHdr
CATFmtNewMeshingRulesEditorHdrAlias
CATFmtFEMRepNumberingHdrAlias
CATFmtNumberingManagerHdrAlias
CATFmtNumberingVisuHdrAlias
CATFmtMeshPartManagerHdrAlias
CATFmtBeamHdrAlias
CATFmtSurfaceTriaRulesMesherHdrAlias
CATFmtOctree2DHdrAlias
CATFmtSurfaceQuadRulesMesherHdrAlias
CATFmt3DRulesMesherHdrAlias
CATFmtOctree3DHdrAlias
CATFmtGHS3DHdrAlias
CATFmtVIDSweep3DMesherHdrAlias
CATFmtHexMesherHdrAlias
CATFmtCoating1DMesherHdrAlias
CATFmtCoating2DMesherHdrAlias
CATFmtHoleFillerMesherHdrAlias
CATFmtPolyhedralMesherHdrAlias
CATFmtTranslationHdrAlias
CATFmtRotationHdrAlias
CATFmtSymmetryHdrAlias
CATFmtOffsetHdrAlias
CATFmtAxisToAxisHdrAlias
CATFmtExtrTranslationHdrAlias
CATFmtExtrRotationHdrAlias
CATFmtExtrSymmetryHdrAlias
CATFmtExtrSpineHdrAlias
CATFmtExtrOffsetMesherHdrAlias
CATFmtInflationMesherHdrAlias
CATFmtMeshingEditionToolsHdrAlias
CATFmtGroupManagerHdrAlias
CATFmtGroupByAssociativityHdrAlias
CATFmtGroupByProximityHdrAlias
CATFmtGroupSpatialHdrAlias
CATFmtGroupBoundaryHdrAlias
CATFmtGroupManualSelectionHdrAlias
CATFmtGroupUnionHdrAlias
CATFmtGroupIntersectionHdrAlias
CATFmtGroupDifferenceHdrAlias
CATFmtContributingPartsHdrAlias
CATFmtContributingFemsHdrAlias
SMAExsUpdateFEMHdrAlias
SMAExsShipStructuresHdrAlias
SMAExsSolidSectionHdrAlias
SMAExsShellSectionHdrAlias
SMAExsContinuumShellSectionHdrAlias
SMAExsCompositeShellSectionHdrAlias
SMAExsCompositeContinuumShellSectionHdrAlias
SMAExsMembraneSectionHdrAlias
SMAExsBeamSectionHdrAlias
SMAExsBeamProfileHdrAlias
SMAExsTrussSectionHdrAlias
SMAExsSPHSectionHdrAlias
SMAExsGasketSectionHdrAlias
SMAExsPretensionSectionHdrAlias
SMAExsRigidBodyConstraintPropertyHdrAlias
SMAExsAnalyticalSurfacePropertyHdrAlias
SMAExsMassRotaryInertiaPropertyHdrAlias
SMAExsRemoteMassRotaryInertiaPropertyHdrAlias

ModelAbstractions	SMAExsMassPerVolumeHdrAlias SMAExsMassPerAreaHdrAlias SMAExsMassPerLengthHdrAlias SMAExsCyclicSymmetryHdrAlias SMAExsConnectionManagerHdrAlias SMAExsConnectionCheckerHdrAlias SMAExsVirtualBoltHdrAlias SMAExsBoltReplicationHdrAlias SMAExsVirtualPartHdrAlias SMAExsSpringHdrAlias SMAExsTieDetectionHdrAlias SMAExsTieHdrAlias SMAExsSpotWeldHdrAlias SMAExsPointFastenerOverrideHdrAlias SMAExsSeamWeldHdrAlias SMAExsLineFastenerOverrideHdrAlias SMAExsImportFastenerHdrAlias SMAExsGeneralConnectionHdrAlias SMAExsRigidConnectionHdrAlias SMAExsVirtualPinHdrAlias SMAExsDisconnectedPartsHdrAlias SMAExsTouchContactHdrAlias SMAExsIntersectContactHdrAlias
ModelConnections	

Description of Commands

The following table provides a description of each of the commands given above.

Caution: If copying the command header ids, be sure to remove the hyphens. The actual header ids do not contain any hyphen.

CommandHeader Id	Description
FEM	
SMAExsSelectFemCmdHdr	Provides to select an existing FEM Rep for an Analysis Case.
SMAExsNewFemCmdHdr	Creates a FEM Rep for an Analysis Case.
SMAExsSwitchToModelerHdr	Switches to the Modeling app.
SMAExsElementTypeAssignmentHdr	Creates a local ElementTypeAssignment feature.
Material	
SMAFeaAssignMaterialCmdHdr	Assign a material via PLM search.
SMAFeaCreateMaterialCmdHdr	Creates a new material.
SMAFeaEditAppliedMatCmdHdr	Edits the simulation properties of a material.
SMAFeaPickerMatCmdHdr	Assign a material via picking an already in-context material.
SMAFeaDeleteAppliedMatCmdHdr	Unassigns a material.
Contact	
SMAExsContactSearchHdr	Identifies contacting surfaces and creates structural surface-based contact between them.
SMAExsContactPairHdr	Creates a structural surface-based contact pair.
SMAExsGeneralContactHdr	Specifies general contact behavior between all components in a model.
SMAExsContactPropertyHdr	Specifies the structural behavior of contacting surfaces for a structural surface-based contact pair.
SMAExsContactInitializationHdr	Adjusts general contact surfaces to improve the solution.
SMAExsContactInterferenceHdr	Controls handling of over-closed surfaces.
SMAExsContactControlsHdr	Defines optional controls for surface-based contact.
SMAExsDisconnectedPartsHdr	Highlights Parts that do not have any connections.
SMAExsTouchContactHdr	Highlights regions in a model where Parts come into contact.
SMAExsIntersectContactHdr	Highlights regions in a model where Parts intersect.
SMAExsChangeFrictionHdr	Change the values of friction properties from step to step.
SMAExsThermalContactSearchHdr	Identifies contacting surfaces and creates thermal surface-based contact between them.
SMAExsThermalContactPairHdr	Creates a thermal surface-based contact pair.
SMAExsThermalContactPropertyHdr	Specifies the thermal behavior of contacting surfaces for a thermal surface-based contact pair.
Steps	
SMAExsStaticStressHdr	Creates a static stress analysis procedure.
SMAExsViscoStressHdr	Creates a quasi-static stress analysis procedure.
SMAExsImplicitDynamicStressHdr	Creates an implicit-dynamic analysis procedure.
SMAExsExplicitDynamicStressHdr	Creates an explicit-dynamic analysis procedure.
SMAExsStaticStressPerturbationHdr	Creates a linear perturbation static stress analysis procedure.
SMAExsFrequencyHdr	Creates a frequency extraction analysis procedure.
SMAExsComplexFrequencyHdr	Creates a complex frequency extraction analysis procedure.
SMAExsHarmonicResponseHdr	Creates a harmonic response analysis procedure.
SMAExsBuckleHdr	Creates a buckling analysis procedure.
SMAExsModalDynamicHdr	Creates a modal dynamic analysis procedure.
SMAExsSteadyStateHeatTransferHdr	Creates a steady-state heat transfer analysis procedure.
SMAExsTransientHeatTransferHdr	Creates a transient heat transfer analysis procedure.
SMAExsEditCurrentStepHdr	Edits the current procedure.

SMAExsDeleteCurrentStepHdr	Deletes the current procedure.
Amplitudes	
SMAExsTabularAmplitudeHdr	Creates an amplitude curve that is defined by a table of values.
SMAExsSmoothStepAmplitudeHdr	Creates an amplitude curve that is defined by smoothly varying curve.
SMAExsPeriodicAmplitudeHdr	Creates an amplitude curve that is defined by a Fourier series.
Initial Conditions	
SMAExsInitialTemperatureHdr	Creates an initial temperature.
SMAExsInitial-TranslationalVelocityHdr	Creates an initial translational velocity.
SMAExsInitial-RotationalVelocityHdr	Creates an initial rotational velocity.
SMAExsVoidRatioHdr	Initial void ratio value for a coupled pore fluid diffusion/stress simulation.
SMAExsMassScalingHdr	Creates a mass scaling.
SMAExsDampingHdr	Creates a damping source.
Boundary Conditions	
SMAExsClampHdr	Creates a clamp.
SMAExsDisplacementRestraintHdr	Creates a displacement restraint.
SMAExsPlanarSymmetryHdr	Creates a planar symmetry restraint.
SMAExsBoltRestraintHdr	Creates a bolt restraint.
SMAExsBallJointHdr	Creates a balljoint restraint.
SMAExsSliderHdr	Creates a slider restraint.
SMAExsHingeHdr	Creates a hinge restraint.
SMAExsCavityPressureHdr	Creates a cavity pressure.
SMAExsConnector-DisplacementRestraintHdr	Creates a connector displacement restraint.
Loads	
SMAExsPressureHdr	Creates a pressure load.
SMAExsBearingLoadHdr	Creates a bearing load.
SMAExsForceHdr	Creates a force load.
SMAExsImportedForceHdr	Creates an imported force load.
SMAExsRemoteForceHdr	Creates a remote force load.
SMAExsTorqueHdr	Creates a torque load.
SMAExsRemoteTorqueHdr	Creates a remote torque load.
SMAExsGravityHdr	Creates a gravity load.
SMAExsInertiaReliefHdr	Creates an inertia relief load.
SMAExsBoltLoadHdr	Creates a bolt force load.
SMAExsBoltTorqueHdr	Creates a bolt torque load.
SMAExsPrescribed-BoltDisplacementHdr	Creates a bolt displacement load.
SMAExsAppliedTranslationHdr	Creates an applied translation load.
SMAExsAppliedRotationHdr	Creates an applied rotation load.
SMAExsApplied-TranslationalVelocityHdr	Creates an applied translational velocity load.
SMAExsApplied-RotationalVelocityHdr	Creates an applied rotational velocity load.
SMAExsApplied-TranslationalAccelerationHdr	Creates an applied translational acceleration load.
SMAExsApplied-RotationalAccelerationHdr	Creates an applied rotational acceleration load.
SMAExsPrescribedTemperatureHdr	Creates a prescribed temperature.
SMAExsLoadSetHdr	Creates a load set.
SMAExsLoadCaseHdr	Creates a load case.
SMAExsCentrifugalForceHdr	Creates a centrifugal force load.
SMAExsRotaryAccelerationLoadHdr	Creates a rotary acceleration load.
SMAExsDisplacementBaseMotionHdr	Creates a displacement motion load.
SMAExsVelocityBaseMotionHdr	Creates a velocity motion load.
SMAExsAccelerationBaseMotionHdr	Creates an acceleration motion load.
SMAExsConnectorForceHdr	Creates a connector force load.
SMAExsConnectorMomentHdr	Creates a connector moment load.
SMAExsConnector-RelativeDisplacementHdr	Creates a connector displacement load.
SMAExsConnectorRelativeRotationHdr	Creates a connector rotation load.
SMAExsSlidingVelocityHdr	Prescribes either a translational velocity along an axis or a rotational velocity about an axis.
Thermal	
SMAExsTemperatureHdr	Creates a temperature condition.
SMAExsHeatFluxHdr	Creates a heat flux condition.
SMAExsFilmConditionHdr	Creates a film condition.
SMAExsVolumetricHeatSourceHdr	Creates a volumetric heat source condition.
SMAExsRadiationAmbientHdr	Creates an ambient radiation condition.
SMAExsRadiationCavityApproxHdr	Creates a cavity radiation condition.
SMAExsBodyforceHdr	Creates a translational load on a body.
Output Requests	
SMAExsFieldOutputRequestHdr	Creates an output request.
Solve	
SMAExsSimulateHdr	Runs the simulation.

SMAExsSimulationExecution-ModelChecksCmdHdr	Checks the model for errors.
SMAExsSimulationExecution-SolverChecksCmdHdr	Performs model and solver checks.
SMAExsSimulationExecution-FullSolveCmdHdr	Submits model for full simulation.
Results	
SMAHvcResultsVisualizationHdr	Toggles viewing the simulation results.
SMAHvcCreateContourPlotHdr	Creates a new contour plot.
SMAHvcCreateSymbolPlotHdr	Creates a new symbol plot.
SMAHvcCreateIsocontourPlotHdr	Creates a new isocontour plot.
SMAHvcCreateColorCodePlotHdr	Creates a new color code plot.
SMAHvcCreateHistoryPlotHdr	Creates an X-Y plot of your history output data.
SMAHvcImportHistoryCurveHdr	Creates a history plot from data imported from a file.
SMAHvcCreateXYFromPathPlotHdr	Creates an XY plot of results extracted along a Path Plot.
SMAHvcCreateSensorHdr	Determines the minimum and maximum values for a selected variable.
SMAHvcCreateResultantSensorHdr	Computes resultant force and moment for the selected support.
SMAHvcCreateFrequencySensorHdr	Creates parameters for each mode of the selected frequency step.
SMAHvcCreateBuckleModeSensorHdr	Creates parameters for each mode of the selected buckle step.
SMAHvcStreamHdr	Creates a family of curves that are tangential to the velocity vector of the flow and show the direction a particle will travel at any point in time.
SMAHvcDisplayGroupInitializeHdr	Configures the options to remove/keep portions of the model from the display.
SMAHvcCreateReportHdr	Creates a report document that summarizes the results of your simulation.
SMAHvcCreateEnvelopeHdr	Create a user-specified coordinate system.
SMAHvcCreateUserCsysHdr	Create a user-specified coordinate system.
SMAHvcCreatePathPlotHdr	Create Path Plot
SMAHvcPlayAnimationHdr	Starts animating the current set of plots.
SMAHvcExtremaToolbarHdr	Highlights where the maximum and minimum values occur.
SMAHvcCutToolbarGPAHdr	Creates a cut through the model to view its interior.
SMAHvcMultiVPFourSquareHdr	Visualize single/multiple plots in four symmetric views configuration.
SMAHvcMultiVPTriLeftHdr	Visualize single/multiple plots having main view on right.
SMAHvcMultiVPTriBottomHdr	Visualize single/multiple plots having main view on top.
SMAHvcMultiVPTwoRowHdr	Visualize single/multiple plots in two rows configuration.
SMAHvcMultiVPTwoColumnHdr	Visualize single/multiple plots in two column configuration.
SMAHvcMultiVPOffHdr	Visualize plot in single view configuration.
SMAHvcMultiVPSettingsHdr	Customize layouts and it's synchronization settings.
SMAHvcGlobalOptionsHdr	Configures the global display settings.
SMAHvcRenderingSettingsHdr	Configures display of geometry associated with the simulation, including products or parts that did not participate in the simulation.
Mesh Setup	
CATFmtNewMeshingRulesEditorHdrAlias	Creates rules for some kinds of meshes.
CATFmtFEMRepNumberingHdrAlias	Assigns ranges of numbers to nodes and elements.
CATFmtNumberingManagerHdrAlias	Manages node and element numbering.
CATFmtNumberingVisuHdrAlias	Displays the numbers associated with nodes and elements.
Mesh Create	
CATFmtMeshPartManagerHdrAlias	Allows modification, update and delete of mesh parts.
CATFmtBeamHdrAlias	Creates a 1D mesh of beam elements.
CATFmtSurfaceTriaRulesMesherHdrAlias	Creates a surface mesh of triangular elements.
CATFmtOctree2DHdrAlias	Creates a 2D mesh of triangular elements.
CATFmtSurfaceQuadRulesMesherHdrAlias	Creates a surface mesh of quadrilateral elements.
CATFmt3DRulesMesherHdrAlias	Creates a 3D mesh of tetrahedron elements.
CATFmtOctree3DHdrAlias	Creates a 3D mesh of tetrahedron elements.
CATFmtGHS3DHdrAlias	Creates a 3D mesh of tetrahedron elements by filling a 2D surface boundary mesh.
CATFmtVIDSweep3DMesherHdrAlias	Creates a 3D mesh by sweeping a surface mesh through a volume.
CATFmtHexMesherHdrAlias	Creates a 3D mesh dominated by hexahedron elements.
CATFmtCoating1DMesherHdrAlias	Creates bar elements from the edges of surface elements.
CATFmtCoating2DMesherHdrAlias	Creates surface elements by extracting the external faces of solid elements.
CATFmtHoleFillerMesherHdrAlias	Creates a 2D radial mesh to cover a hole or for use as the fluid boundary in an open container.
CATFmtPolyhedralMesherHdrAlias	Creates a polyhedral mesh.
Mesh Operate	
CATFmtTranslationHdrAlias	Creates a mesh by copying and translating an existing mesh.
CATFmtRotationHdrAlias	Creates a mesh by copying and rotating an existing mesh.
CATFmtSymmetryHdrAlias	Creates a mesh by copying an existing mesh through a symmetry plane.
CATFmtOffsetHdrAlias	Creates a mesh by copying and offsetting an existing mesh.
CATFmtAxisToAxisHdrAlias	Creates a mesh by copying and moving an existing mesh from one axis system to another.
CATFmtExtrTranslationHdrAlias	Creates a mesh by extruding and translating nodes or surface elements.
CATFmtExtrRotationHdrAlias	Creates a mesh by extruding and rotating nodes or surface elements.
CATFmtExtrSymmetryHdrAlias	Creates a mesh by extruding nodes or surface elements through a symmetry plane.
CATFmtExtrSpineHdrAlias	Creates a mesh by extruding nodes or surface elements along an edge.
CATFmtExtrOffsetMesherHdrAlias	Creates a mesh by extruding and offsetting nodes or surface elements.
CATFmtInflationMesherHdrAlias	Creates a 3D mesh from a 2D mesh.
CATFmtMeshingEditionToolsHdrAlias	Accesses the manual editing tools.

Mesh Group

CATFmtGroupManagerHdrAlias	Allows modification, update and delete of groups.
CATFmtGroupByAssociativityHdrAlias	Creates a group containing all the mesh entities in a selected region.
CATFmtGroupByProximityHdrAlias	Creates a group of mesh entities by proximity.
CATFmtGroupSpatialHdrAlias	Creates a group of mesh entities that lie within a bounding box or sphere.
CATFmtGroupByBoundaryHdrAlias	Creates a group of mesh entities that lie within a boundary.
CATFmtGroupByManualSelectionHdrAlias	Creates a group of mesh entities using a manual selection process.
CATFmtGroupUnionHdrAlias	Creates a group of mesh entities from the union of existing groups.
CATFmtGroupIntersectionHdrAlias	Creates a group of mesh entities from the intersection of existing groups.
CATFmtGroupDifferenceHdrAlias	Creates a group of mesh entities from the difference between two groups.

Manage Model

CATFmtContributingPartsHdrAlias	Manages inclusion of 3D geometric entities.
CATFmtContributingFemsHdrAlias	Manages inclusion of component finite element models.
SMAExsUpdateFEMHdrAlias	Checks the finite element model.
SMAExsShipStructuresHdrAlias	Creates a functional structure finite element model.

Sections

SMAExsSolidSectionHdrAlias	Creates a solid section.
SMAExsShellSectionHdrAlias	Creates a shell section.
SMAExsContinuumShellSectionHdrAlias	Creates a continuum shell section.
SMAExsCompositeShellSectionHdrAlias	Creates a composite shell section.
SMAExsCompositeContinuumShellSectionHdrAlias	Creates a composite continuum shell section.
SMAExsMembraneSectionHdrAlias	Creates a membrane section.
SMAExsBeamSectionHdrAlias	Creates a beam section.
SMAExsBeamProfileHdrAlias	Defines a beam profile.
SMAExsTrussSectionHdrAlias	Creates a truss section.
SMAExsSPHSectionHdrAlias	Converts a Lagrangian finite element mesh to smoothed particles.
SMAExsGasketSectionHdrAlias	Creates a gasket.
SMAExsPretensionSectionHdrAlias	Defines a planar surface through the body of a fastener through which bolt loads and restraints are applied.

Abstractions

SMAExsRigidBodyConstraintPropertyHdrAlias	Creates a rigid body.
SMAExsAnalyticalSurfacePropertyHdrAlias	Creates a rigid surface.
SMAExsMassRotaryInertiaPropertyHdrAlias	Applies nonstructural mass and inertia on a point.
SMAExsRemoteMassRotaryInertiaPropertyHdrAlias	Applies remote mass and inertia.
SMAExsMassPerVolumeHdrAlias	Applies nonstructural mass on a body.
SMAExsMassPerAreaHdrAlias	Applies nonstructural mass on a face.
SMAExsMassPerLengthHdrAlias	Applies nonstructural mass along a beam.
SMAExsCyclicSymmetryHdrAlias	Creates a cyclic symmetry model.

Connections

SMAExsConnectionManagerHdrAlias	Manages connections.
SMAExsConnectionCheckerHdrAlias	Checks connection validity.
SMAExsVirtualBoltHdrAlias	Creates a virtual bolt.
SMAExsBoltReplicationHdrAlias	Identifies and replicates bolts between parts.
SMAExsVirtualPartHdrAlias	Creates a virtual part.
SMAExsSpringHdrAlias	Creates a spring.
SMAExsTieDetectionHdrAlias	Identifies contacting surfaces and creates tie connections between them.
SMAExsTieHdrAlias	Creates a tie connection.
SMAExsSpotWeldHdrAlias	Creates a spot weld.
SMAExsPointFastenerOverrideHdrAlias	Overrides a point fastener's construct.
SMAExsSeamWeldHdrAlias	Creates a seam weld.
SMAExsLineFastenerOverrideHdrAlias	Overrides a line fastener's construct.
SMAExsImportFastenerHdrAlias	Imports point/line fasteners.
SMAExsGeneralConnectionHdrAlias	Creates a general connection.
SMAExsRigidConnectionHdrAlias	Creates a rigid connection.
SMAExsVirtualPinHdrAlias	Creates a virtual pin.
SMAExsDisconnectedPartsHdrAlias	Highlights parts that do not have any connections.
SMAExsTouchContactHdrAlias	Highlights regions in a model where parts come into contact but do not intersect or overlap.
SMAExsIntersectContactHdrAlias	Highlights regions in a model where parts intersect or overlap.

Caution: If copying the above command header ids, be sure to remove the hyphens. The actual header ids do not contain any hyphen.

History

Version: 1 [Jan 2016] Document created

Discovering Command Header Identifiers

Technical Article

Abstract

This article details how to use the **Application Frame Structure Exposition** command to obtain the command header identifier (for non-Simulation app commands) to use for the **headerId** attribute of the **<CommandHeader>** element.

The **Application Frame Structure Exposition** command generates a file that contains a listing of all the commands available in the selected context. This listing gives the internal command header identifier, shown as 'Id', for each command. This is the value to use for the **headerId** attribute.

To run the **Application Frame Structure Exposition** command:

- Within the **3DEXPERIENCE** desktop app,
 - Display the Status Bar (displays at the bottom of the window).
 - In the Power Input field (right side of Status Bar), enter

c:Application Frame Structure Exposition

On the **Structure Exposition** dialog,

- enter the desired target directory,
- select the desired Workshop/Workbench in the list,
- hit the Print button to have a file written to the given directory.

The **Application Frame Structure Exposition** command will reveal only the commands that are available in the current context. Thus, to discover a command that is available in a particular app, the **Application Frame Structure Exposition** command must be run within that app.

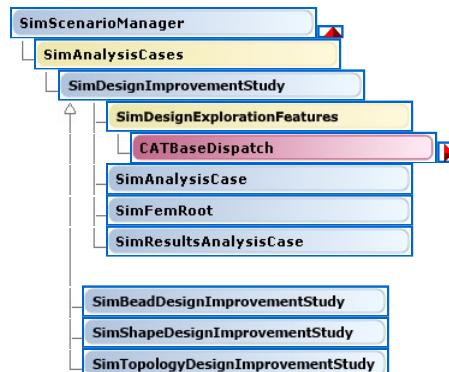
The **Application Frame Structure Exposition** command may reveal commands that do not work with the custom Method <CommandHeader> element. This is a limitation of the command itself and cannot be resolved by the custom Method. Nevertheless, in such a case be sure to double check the spelling of the command header id.

History

Version: 1 [Jan 2016] Document created

Multiphysics Scenario Creation Object Model Map

See Also [Legend](#)



Use the **GetItem** method of the **SimSpecsRepManager** object to retrieve the **SimScenarioManager** object.

VBA Python

```

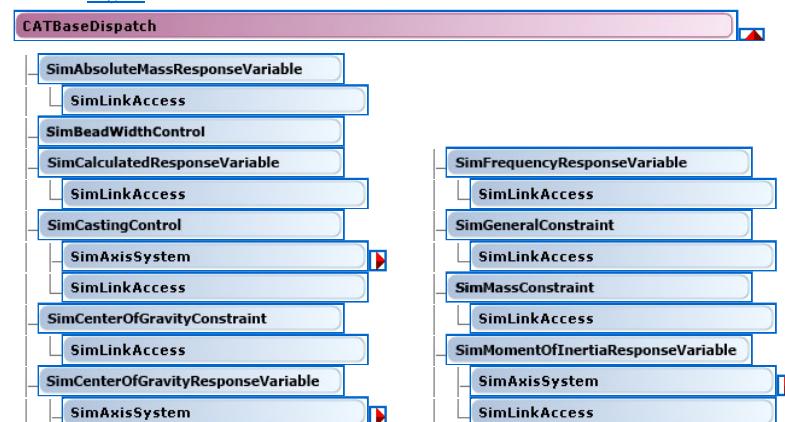
Dim oSimSpecsRepManager As SimSpecsRepManager
...
Dim oSimScenarioManager As SimScenarioManager
Set oSimScenarioManager = oSimSpecsRepManager.GetItem("SimScenarioManager")

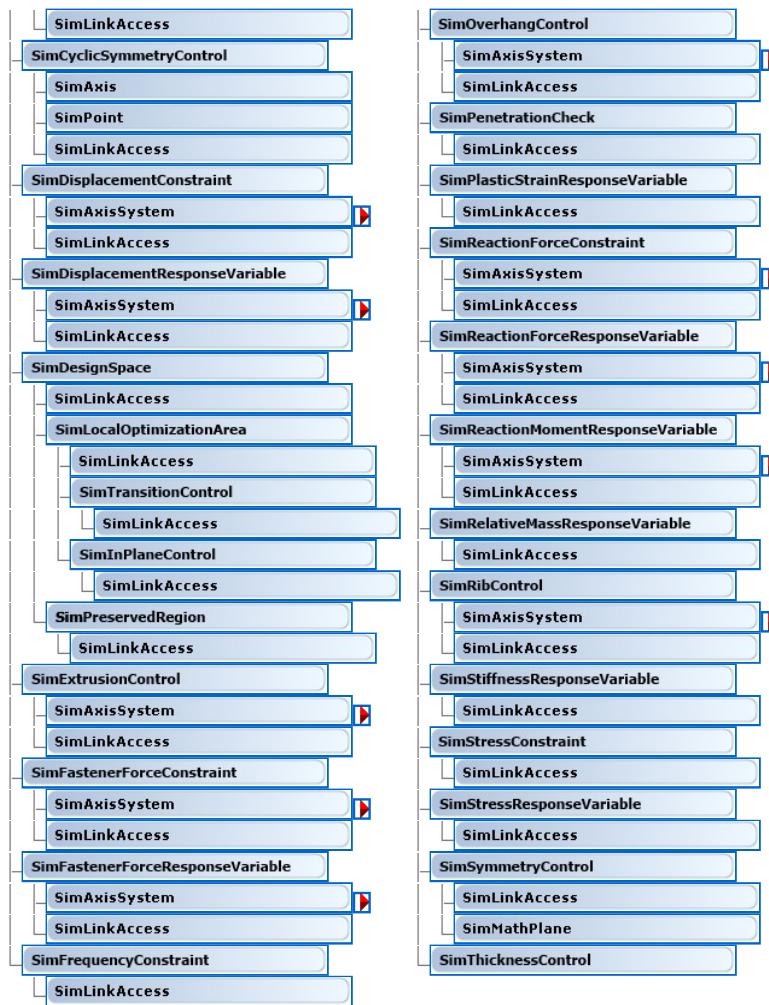
# Get the simulation editor and the simulation root
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
MySimulationRoot = myEntities.Item(1)

# Retrieve the Scenario Manager
myScenarioManager = MySimulationRoot.GetItem("SimScenarioManager")
  
```

Simulation Feature Objects

See Also [Legend](#)





♦ Available with the *RBM - Simulation Scripting* product only.

Use the **Features** property of the **SimDesignImprovementStudyCase** object to retrieve the **SimDesignExplorationFeatures** collection.

VBA Python

```

Dim oSimDesignImprovementStudyCase As SimDesignImprovementStudyCase
...
Dim cSimDesignExplorationFeatures As SimDesignExplorationFeatures
Set cSimDesignExplorationFeatures = oSimDesignImprovementStudyCase.Features

...
cSimDesignExplorationFeatures = oSimDesignImprovementStudyCase.Features
...

```

Use the **Add** method of the **SimDesignExplorationFeatures** collection to create a new simulation feature, such as a **SimDisplacementConstraint** object.

VBA Python

```

Dim oSimDisplacementConstraint As SimDisplacementConstraint
Set oSimDisplacementConstraint = cSimDesignExplorationFeatures.Add("SimDisplacementConstraint")

...
oSimDisplacementConstraint = cSimDesignExplorationFeatures.Add("SimDisplacementConstraint")
...

```

Use the **AddSubFeature** method of the **SimDesignExplorationFeatures** collection to create a new simulation feature under an aggregating object, such as a **SimPreservedRegion** object.

VBA Python

```

Dim oSimDesignSpaceAs SimDesignSpace
Set oSimDesignSpace= cSimDesignExplorationFeatures.Add("SimDesignSpace")

Dim oSimPreservedRegion As SimPreservedRegion
Set oSimPreservedRegion = cSimDesignExplorationFeatures.AddSubFeature("SimPreservedRegion",oDesignSpace)

...
oSimPreservedRegion = oSimPreservedRegion .AddSubFeature("SimPreservedRegion",oDesignSpace)
...

```

See [Creation Late Types for Design Exploration Scenario Features](#) to know how you can create the simulation features.

Use the **TypeName** function to determine the type of an object retrieved from the **SimDesignExplorationFeatures** collection in order to use its own properties and methods.

VBA Python

```
Dim oRetrievedObject As AnyObject
Set oRetrievedObject = cSimDesignExplorationFeatures.Item(3)
Dim objectType As String
objectType = TypeName(oRetrievedObject)

...
oRetrievedObject = cSimFeatures.Item(3)
objectType = TypeName(oRetrievedObject)
...
```

Use the **GetItem** method to return the **SimLinkAccess** object from its aggregating object, such as the **SimStressConstraint** object.

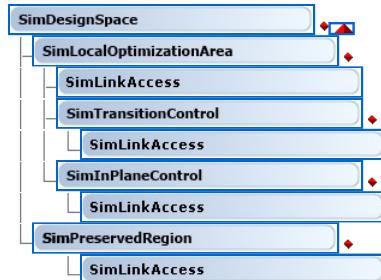
VBA Python

```
Dim oSimStressConstraintAs SimStressConstraint
...
Dim oSimLinkAccess As SimLinkAccess
Set oSimLinkAccess = oSimStressConstraint.GetItem("SimLinkAccess")

...
oSImLinkAccess = oSimStressConstraint.GetItem("SimLinkAccess")
...
```

SimDesignSpace Object

See Also [Legend](#)



◆ Available with the *RBM - Simulation Scripting* product only.

Use the **Add** method of the **SimDesignExplorationFeatures** collection to create a new **SimDesignSpace** object.

VBA Python

```
Dim cSimDesignExplorationFeatures As SimDesignExplorationFeatures
...
Dim oSimDesignSpace As SimDesignSpace
Set oSimDesignSpace = cSimFeatures.Add("SimDesignSpace")

...
oSImDesignSpace = cSimDesignExplorationFeatures.Add("SimDesignSpace")
...
```

Creating and Executing a Topology Design Improvement Study

This article explains how to create and execute a topology design improvement study.

Before you begin:

- Launch CATIA and then import and open the `CAAScdfeaBracketModel.3dxml` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaDesignExplorationScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Make sure that you have all of the licenses required for launching execution and perform the solve operation in the interactive session for a similar simulation to test the licenses.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaTopologyDesignImprovementStudySource.htm](#)

This use case can be divided in 15 steps:

1. [Retrieving the active Product](#)
2. [Retrieving the Publications and Composing Links](#)
3. [Creating the FEM Rep and Retrieving its Rep Manager](#)
4. [Creating a Mesh on a Part](#)
5. [Creating a Property](#)
6. [Initializing and Creating the Design Exploration PLM Object](#)
7. [Retrieving the Scenario Manager](#)
8. [Creating the Structural Analysis Case with the FEM Rep](#)
9. [Creating and Initializing the Static Step](#)
10. [Retrieving the Structural Analysis Case Features Set](#)
11. [Creating Structural Analysis Case features](#)
12. [Creating the Topology Design Improvement Study with the Structural Analysis Case](#)

13. [Retrieving the Design Improvement Study features](#)
14. [Creating the Design Improvement Study features](#)
15. [Launching the Shape Design Improvement Study](#)

1. Retrieving the active Product

As a first step, the UC retrieves the active product.

```
...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor

Dim myProdService As PLMProductService
Set myProdService = myEditor.GetService("PLMProductService")

Dim myEntities As PLMEntities
Set myEntities = myProdService.EditedContent

Dim myRootProduct As VPMReference
Set myRootProduct = myEntities.Item(1)
...
```

2. Retrieving the Publications and Composing Links

In this step UC retrieves the publications aggregated by the root product and composes links.

```
...
Dim myPublications As VPMPublications
Set myPublications = myRootProduct.Publications

Dim myDesignSpaceSupport As VPMPublication
Set myDesignSpaceSupport = myPublications.GetItem("Design_Space")

' Composing link to the publication

Dim myRootOccurrence As VPMRootOccurrence
Set myRootOccurrence = myProdService.RootOccurrence

Dim myRepInstance As VPMRepInstance
Set myRepInstance = Nothing ' Because we use publications

Dim myLinkToPartitionSupport As AnyObject
Set myLinkToPartitionSupport = myProdService.ComposeLink(myRootOccurrence, myRepInstance, myDesignSpaceSupport)

...
```

3. Creating the FEM Rep and Retrieving its Rep Manager

In this step UC creates a finite element representation model on the product which was just opened.

For further information about creating the FEM Rep and retrieving its Rep Manager, refer to the article [Creating FEM representation](#).

4. Creating a Mesh on a Part

In this step UC retrieves the nodes and elements set from the FEM Representation, creates a mesh on a part and sets the mesh attributes.

For further information about the creation and management of mesh parts, refer to the article [Creating Mesh Part](#).

5. Creating a Property

In this step UC creates a solid section on the part and sets its individual attributes.

For further information about the creation and management of properties, refer to the article [Managing FEM section features](#).

6. Initializing and Creating the Design Exploration Simulation PLM Object

In this step UC creates a simulation object.

```
...
' Initializing the simulation PLM Object

Dim myInitService As SimInitializationService
Set myInitService = CATIA.GetSessionService("SimInitializationService")

myInitService.SetSimulationInitialization "SimDesignexplorationCreation", "SimOptimization"

' Creating the simulation PLM Object

Dim mySimPLMService As SIMPLMService
Set mySimPLMService = CATIA.GetSessionService("SIMPLMService")

Dim mySimulationReference As SimulationReference
mySimPLMService.PLCreate "NewDesignExplorationObject", "SMAfeaPLMNewSimu", myModel, mySimulationReference, myEditor
...
```

For further information about the initialization of a simulation object, refer to the article [Initializing a Simulation Object](#).

7. Retrieving the Scenario Manager

In this step UC retrieves the scenario manager.

```
...
Dim myScenarioManager As SimScenarioManager
Set myScenarioManager = mySimulationReference.GetItem("SimScenarioManager")
...
```

8. Creating the Structural Analysis Case with the FEM Rep

In this step UC creates a structural analysis case and associates it with the FEM rep.

For further information about the creation and management of structural analysis case, refer to the article [Creating and Executing a Simulation](#).

9. Creating and Initializing the Static Step

In this step UC creates a static step under the structural analysis case and sets its individual attributes.

For further information about the creation and management of steps, refer to the article [Creating and Executing a Simulation](#).

10. Retrieving the Structural Analysis Case Features Set

In this step UC retrieves the features set for the structural analysis case.

For further information about the management of structural analysis case features set, refer to the article [Creating and Executing a Simulation](#).

11. Creating Structural Analysis Case features

In this step UC creates and initializes features for the structural analysis case.

For further information about the creation and management of structural analysis features, refer to the article [Creating and Executing a Simulation](#).

12. Creating the Topology Design Improvement Study with the Structural Analysis case

In this step UC creates a Topology Design Improvement Study and associates it with the structural analysis case.

```
...
Dim myAnalysisCases As SimAnalysisCases
Set myAnalysisCases = myScenarioManager.AnalysisCases

Dim myTopologyStudy As SimTopologyDesignImprovementStudy
Set myTopologyStudy = myAnalysisCases.AddAnalysisCase("SimTopologyDesignImprovementStudy")

myTopologyStudy.AnalysisCase = myAnalysisCase
...
```

13. Retrieving the Design Improvement Study Features Set

In this step UC retrieves the design exploration features set.

```
...
Dim myTopologyFeatures As SimDesignImprovementFeatures
Set myTopologyFeatures = myTopologyStudy.Features
...
```

14. Creating Design Improvement Study Features

In this step UC creates features for the simulation.

```
...
Dim myDesignSpace As SimDesignSpace
Set myDesignSpace = myFeatures.Add("SimDesignSpace")

' Setting feature support

Dim myLinkAccess As SimLinkAccess
Set myLinkAccess = myDesignSpace.GetItem("SimLinkAccess")
myLinkAccess.AddLink "MainSupport", myLinkToPartitionSupport
...
```

You can find the available late types for features creation in the [Creation Late Types for Design Exploration Scenario Features](#) article.

15. Launching the Simulation

In this step UC launch the topology design improvement study and shows the results.

```
...
Dim myExecutionService As SimExecutionService
Set myExecutionService = myEditor.GetService("SimExecutionService")

myExecutionService.BasicExecuteAll mySimulationReference

' switching to the results visualization workbench
CATIA.StartWorkbench "SMAHvcResultsVisualizationWorkbench"
...
```

Initializing and Executing a Topology Design Improvement Study

This article explains how to initialize and execute a topology design improvement study.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaBracketModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaDesignExplorationScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Make sure that you have all of the licenses required for launching execution and perform the solve operation in the interactive session for a similar simulation to test the licenses.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaTopologyDesignImprovementStudyWithPythonSource.htm](#)

This use case can be divided in 12 steps:

1. [Retrieving the active Product](#)
2. [Retrieving the Publications and Composing Links](#)
3. [Creating the FEM Rep and Retrieving its Rep Manager](#)
4. [Creating a Mesh on a Part](#)
5. [Creating a Property](#)
6. [Initializing and Creating the Design Exploration PLM Object](#)
7. [Retrieving the Scenario Manager](#)
8. [Creating the Structural Analysis Case with the FEM Rep](#)
9. [Creating and Initializing the Static Step](#)
10. [Retrieving the Structural Analysis Case Features Set](#)
11. [Creating Structural Analysis Case features](#)
12. [Creating the Topology Design Improvement Study with the Structural Analysis Case](#)
13. [Retrieving the Design Improvement Study features](#)
14. [Creating the Design Improvement Study features](#)
15. [Launching the topology design improvement study](#)

1. Retrieves the product

As a first step, the UC retrieves the active product

```
...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
...
```

2. Retrieves the Publications and Composes Links

In this step UC retrieves the publications aggregated by the root product and composes links.

```
...
myPublications = myModel.Publications
myDesignSpaceSupport = myPublications.GetItem("Design_Space")

# Composing links
myLinkToPartitionSupport = myProdService.ComposeLink(None, None, myDesignSpaceSupport)
...
```

As we use publication, we just need to give it as third argument to compose the link

3. Creates the FEM Rep and Retrieving its Rep Manager

In this step UC creates a finite element representation model on the product which was just opened.

For further information about creating the FEM Rep and retrieving its Rep Manager, refer to the article [Creating FEM representation](#).

4. Creates a Mesh on a Part

In this step UC retrieves the nodes and elements set from the FEM Representation, creates a mesh on a part and sets the mesh attributes.

For further information about the creation and management of mesh parts, refer to the article [Creating Mesh Part](#).

5. Creates a Property

In this step UC creates a solid section on the part and sets its individual attributes.

For further information about the creation and management of properties, refer to the article [Managing FEM section features](#).

6. Initializing and Creating the Design Exploration Simulation PLM Object

In this step UC creates a simulation object.

```
...
# Creating and Initializing the design exploration PLM Object
myInitService = CATIA.GetService("SimInitializationService")
myInitService.SetSimulationInitialization "SimDesignexplorationCreation", "SimOptimization"
mySimPLMService = CATIA.GetService("SIMPLMService")
mySimPLMService.PLMCreat "NewDesignExplorationObject", "SMAfeaPLMNewSimu", myModel, mySimulationReference, myEditor
...
```

For further information about the initialization of a simulation object, refer to the article [Initializing a Simulation Object](#).

7. Retrieves the Scenario Manager

In this step UC retrieves the scenario manager.

```
...
myScenarioManager = MySimulationRoot.GetItem("SimScenarioManager")
```

...

8. Creating the Structural Analysis Case with the FEM Rep

In this step UC retrieves the structural analysis case and associates it with the FEM rep.

For further information about the creation and management of structural analysis case, refer to the article [Creating and Executing a Simulation \(Python\)](#).

9. Creating and Initializing the Static Step

In this step UC creates a static step under the structural analysis case and sets its individual attributes.

For further information about the creation and management of steps, refer to the article [Creating and Executing a Simulation \(Python\)](#).

10. Retrieving the Structural Analysis Case Features Set

In this step UC retrieves the features set for the structural analysis case.

For further information about the management of structural analysis case features set, refer to the article [Creating and Executing a Simulation \(Python\)](#).

11. Creating Structural Analysis Case features

In this step UC creates and initializes features for the structural analysis case.

For further information about the creation and management of structural analysis features, refer to the article [Creating and Executing a Simulation \(Python\)](#).

12. Associates the Topology Design Improvement Study with the Structural Analysis Case

In this step UC retrieves the topology design improvement study and associates it with the structural analysis case

```
...
myAnalysisCases = myScenarioManager.AnalysisCases
myTopologyStudy = myAnalysisCases.AddAnalysisCase("SimTopologyDesignImprovementStudy")
myTopologyStudy.AnalysisCase = myAnalysisCase
...
```

13. Retrieving the Design Improvement Study Features Set

In this step UC retrieves the features set.

```
...
myTopologyFeatures = myTopologyStudy.Features
...
```

14. Creating Design Improvement Study Features

In this step UC creates features for the topology design improvement study.

```
...
# Creating Design Space feature and setting its support
myDesignSpace = myFeatures.Add("SimDesignSpace")
myLinkAccess = myDesignSpace .GetItem("SimLinkAccess")
myLinkAccess.AddLink "MainSupport", myLinkToPartitionSupport
...
```

You can find the available late types for features creation in the [Creation Late Types for Design Exploration Scenario Features](#) article.

15. Launches the Topology Design Improvement Study

In this step UC launch the topology design improvement study and shows the results.

```
...
myExecutionService = myEditor.GetService("SimExecutionService")
myExecutionService.BasicExecuteAll (mySimulationReference)
...
```

Creating and Executing a Shape Design Improvement Study

This article explains how to create and execute a shape design improvement study.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaCarrierModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaDesignExplorationScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Make sure that you have all of the licenses required for launching execution and perform the solve operation in the interactive session for a similar simulation to test the licenses.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaShapeDesignImprovementStudySource.htm](#)

This use case can be divided in 15 steps:

1. [Retrieving the active Product](#)
2. [Retrieving the Publications and Composing Links](#)
3. [Creating the FEM Rep and Retrieving its Rep Manager](#)
4. [Creating a Mesh on a Part](#)

5. [Creating a Property](#)
6. [Initializing and Creating the Design Exploration PLM Object](#)
7. [Retrieving the Scenario Manager](#)
8. [Creating the Structural Analysis Case with the FEM Rep](#)
9. [Creating and Initializing the Static Step](#)
10. [Retrieving the Structural Analysis Case Features Set](#)
11. [Creating Structural Analysis Case features](#)
12. [Creating the Shape Design Improvement Study with the Structural Analysis Case](#)
13. [Retrieving the Design Improvement Study features](#)
14. [Creating the Design Improvement Study features](#)
15. [Launching the Shape Design Improvement Study](#)

1. Retrieving the active Product

As a first step, the UC retrieves the active product.

```
...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor

Dim myProdService As PLMProductService
Set myProdService = myEditor.GetService("PLMProductService")

Dim myEntities As PLMEntities
Set myEntities = myProdService.EditedContent

Dim myRootProduct As VPMReference
Set myRootProduct = myEntities.Item(1)
...
```

2. Retrieving the Publications and Composing Links

In this step UC retrieves the publications aggregated by the root product and composes links.

```
...
Dim myPublications As VPMPublications
Set myPublications = myRootProduct.Publications

Dim myDesignSpaceSupport As VPMPublication
Set myDesignSpaceSupport = myPublications.GetItem("Design_Space")

' Composing link to the publication

Dim myRootOccurrence As VPMRootOccurrence
Set myRootOccurrence = myProdService.RootOccurrence

Dim myRepInstance As VPMRepInstance
Set myRepInstance = Nothing ' Because we use publications

Dim myLinkToDisconnectSupport As AnyObject
Set myLinkToDisconnectSupport = myProdService.ComposeLink(myRootOccurrence, myRepInstance, myDesignSpaceSupport)

...
```

3. Creating the FEM Rep and Retrieving its Rep Manager

In this step UC creates a finite element representation model on the product which was just opened.

For further information about creating the FEM Rep and retrieving its Rep Manager, refer to the article [Creating FEM representation](#).

4. Creating a Mesh on a Part

In this step UC retrieves the nodes and elements set from the FEM Representation, creates a mesh on a part and sets the mesh attributes.

For further information about the creation and management of mesh parts, refer to the article [Creating Mesh Part](#).

5. Creating a Property

In this step UC creates a solid section on the part and sets its individual attributes.

For further information about the creation and management of properties, refer to the article [Managing FEM section features](#).

6. Initializing and Creating the Design Exploration Simulation PLM Object

In this step UC creates a simulation object.

```
...
' Initializing the simulation PLM Object

Dim myInitService As SimInitializationService
Set myInitService = CATIA.GetSessionService("SimInitializationService")

myInitService.SetSimulationInitialization "SimDesignexplorationCreation", "SimOptimization"

' Creating the simulation PLM Object

Dim mySimPLMService As SIMPLMService
Set mySimPLMService = CATIA.GetSessionService("SIMPLMService")

Dim mySimulationReference As SimulationReference
mySimPLMService.PLCreate "NewDesignExplorationObject", "SMAfeaPLMNewSimu", myModel, mySimulationReference, myEditor
...
```

For further information about the initialization of a simulation object, refer to the article [Initializing a Simulation Object](#).

7. Retrieving the Scenario Manager

In this step UC retrieves the scenario manager.

```
...
Dim myScenarioManager As SimScenarioManager
Set myScenarioManager = mySimulationReference.GetItem("SimScenarioManager")
...
```

8. Creating the Structural Analysis Case with the FEM Rep

In this step UC creates a structural analysis case and associates it with the FEM rep.

For further information about the creation and management of structural analysis case, refer to the article [Creating and Executing a Simulation](#).

9. Creating and Initializing the Static Step

In this step UC creates a static step under the structural analysis case and sets its individual attributes.

For further information about the creation and management of steps, refer to the article [Creating and Executing a Simulation](#).

10. Retrieving the Structural Analysis Case Features Set

In this step UC retrieves the features set for the structural analysis case.

For further information about the management of structural analysis case features set, refer to the article [Creating and Executing a Simulation](#).

11. Creating Structural Analysis Case features

In this step UC creates and initializes features for the structural analysis case.

For further information about the creation and management of structural analysis features, refer to the article [Creating and Executing a Simulation](#).

12. Creating the Shape Design Improvement Study with the Structural Analysis case

In this step UC creates a Shape Design Improvement Study and associates it with the structural analysis case.

```
...
Dim myAnalysisCases As SimAnalysisCases
Set myAnalysisCases = myScenarioManager.AnalysisCases

Dim myShapeStudy As SimShapeDesignImprovementStudy
Set myShapeStudy= myAnalysisCases.AddAnalysisCase("SimShapeDesignImprovementStudy")

myShapeStudy.AnalysisCase = myAnalysisCase
...
```

13. Retrieving the Design Improvement Study Features Set

In this step UC retrieves the design exploration features set.

```
...
Dim myShapeFeatures As SimDesignImprovementFeatures
Set myShapeFeatures = myShapeFeatures.Features
...
```

14. Creating Design Improvement Study Features

In this step UC creates features for the simulation.

```
...
Dim myDesignSpace As SimDesignSpace
Set myDesignSpace = myFeatures.Add("SimDesignSpace")

' Setting feature support

Dim myLinkAccess As SimLinkAccess
Set myLinkAccess = myDesignSpace .GetItem("SimLinkAccess")
myLinkAccess.AddLink "MainSupport", myLinkToDisconnectSupport
...
```

You can find the available late types for features creation in the [Creation Late Types for Design Exploration Scenario Features](#) article.

15. Launching the Shape Design Improvement Study

In this step UC launch the topology design improvement study and shows the results.

```
...
Dim myExecutionService As SimExecutionService
Set myExecutionService = myEditor.GetService("SimExecutionService")

myExecutionService.BasicExecuteAll mySimulationReference

' switching to the results visualization workbench
CATIA.StartWorkbench "SMAHvcResultsVisualizationWorkbench"
...
```

Initializing and Executing a Shape Design Improvement Study

This article explains how to initialize and execute a shape design improvement study.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaCarrierModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaDesignExplorationScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Make sure that you have all of the licenses required for launching execution and perform the solve operation in the interactive session for a similar simulation to test the licenses.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaShapeDesignImprovementStudyWithPythonSource.htm](#)

This use case can be divided in 12 steps:

1. [Retrieving the active Product](#)
2. [Retrieving the Publications and Composing Links](#)
3. [Creating the FEM Rep and Retrieving its Rep Manager](#)
4. [Creating a Mesh on a Part](#)
5. [Creating a Property](#)
6. [Initializing and Creating the Design Exploration PLM Object](#)
7. [Retrieving the Scenario Manager](#)
8. [Creating the Structural Analysis Case with the FEM Rep](#)
9. [Creating and Initializing the Static Step](#)
10. [Retrieving the Structural Analysis Case Features Set](#)
11. [Creating Structural Analysis Case features](#)
12. [Creating the Shape Design Improvement Study with the Structural Analysis Case](#)
13. [Retrieving the Design Improvement Study features](#)
14. [Creating the Design Improvement Study features](#)
15. [Launching the shape design improvement study](#)

1. Retrieves the active product

As a first step, the UC retrieves the active product

```
...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
...
```

2. Retrieves the Publications and Composes Links

In this step UC retrieves the publications aggregated by the root product and composes links.

```
...
myPublications = myModel.Publications
myDesignSpaceSupport = myPublications.GetItem("Design_Space")

# Composing links
myLinkToDisconnectSupport = myProdService.ComposeLink(None, None, myDesignSpaceSupport)
...
```

As we use publication, we just need to give it as third argument to compose the link

3. Creates the FEM Rep and Retrieving its Rep Manager

In this step UC creates a finite element representation model on the product which was just opened.

For further information about creating the FEM Rep and retrieving its Rep Manager, refer to the article [Creating FEM representation](#).

4. Creates a Mesh on a Part

In this step UC retrieves the nodes and elements set from the FEM Representation, creates a mesh on a part and sets the mesh attributes.

For further information about the creation and management of mesh parts, refer to the article [Creating Mesh Part](#).

5. Creates a Property

In this step UC creates a solid section on the part and sets its individual attributes.

For further information about the creation and management of properties, refer to the article [Managing FEM section features](#).

6. Initializing and Creating the Design Exploration Simulation PLM Object

In this step UC creates a simulation object.

```
...
# Creating and Initializing the design exploration PLM Object
myInitService = CATIA.GetService("SimInitializationService")
myInitService.SetSimulationInitialization "SimDesignExplorationCreation", "SimOptimization"
mySimPLMService = CATIA.GetService("SIMPLMService")
mySimPLMService.PLMSCreate "NewDesignExplorationObject", "SMAFeaPLMNewSimu", myModel, mySimulationReference, myEditor
...
```

For further information about the initialization of a simulation object, refer to the article [Initializing a Simulation Object](#).

7. Retrieves the Scenario Manager

In this step UC retrieves the scenario manager.

```
...
myScenarioManager = MySimulationRoot.GetItem("SimScenarioManager")
```

...

8. Creating the Structural Analysis Case with the FEM Rep

In this step UC retrieves the structural analysis case and associates it with the FEM rep.

For further information about the creation and management of structural analysis case, refer to the article [Creating and Executing a Simulation \(Python\)](#).

9. Creating and Initializing the Static Step

In this step UC creates a static step under the structural analysis case and sets its individual attributes.

For further information about the creation and management of steps, refer to the article [Creating and Executing a Simulation \(Python\)](#).

10. Retrieving the Structural Analysis Case Features Set

In this step UC retrieves the features set for the structural analysis case.

For further information about the management of structural analysis case features set, refer to the article [Creating and Executing a Simulation \(Python\)](#).

11. Creating Structural Analysis Case features

In this step UC creates and initializes features for the structural analysis case.

For further information about the creation and management of structural analysis features, refer to the article [Creating and Executing a Simulation \(Python\)](#).

12. Associates the Shape Design Improvement Study with the Structural Analysis Case

In this step UC retrieves the Shape design improvement study and associates it with the structural analysis case

```
...
myAnalysisCases = myScenarioManager.AnalysisCases
myShapeStudy = myAnalysisCases.AddAnalysisCase("SimShapeDesignImprovementStudy")
myShapeStudy.AnalysisCase = myAnalysisCase
...
```

13. Retrieving the Design Improvement Study Features Set

In this step UC retrieves the features set.

```
...
myShapeFeatures = myShapeStudy.Features
...
```

14. Creating Design Improvement Study Features

In this step UC creates features for the Shape design improvement study.

```
...
# Creating Design Space feature and setting its support
myDesignSpace = myFeatures.Add("SimDesignSpace")
myLinkAccess = myDesignSpace .GetItem("SimLinkAccess")
myLinkAccess.AddLink "MainSupport", myLinkToDisconnectSupport
...
```

You can find the available late types for features creation in the [Creation Late Types for Design Exploration Scenario Features](#) article.

15. Launches the Shape Design Improvement Study

In this step UC launch the Shape design improvement study and shows the results.

```
...
myExecutionService = myEditor.GetService("SimExecutionService")
myExecutionService.BasicExecuteAll (mySimulationReference)
...
```

Creating and Executing a Bead Design Improvement Study

This article explains how to create and execute a Bead design improvement study.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaClampModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaDesignExplorationScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Make sure that you have all of the licenses required for launching execution and perform the solve operation in the interactive session for a similar simulation to test the licenses.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaBeadDesignImprovementStudySource.htm](#)

This use case can be divided in 15 steps:

1. [Retrieving the active Product](#)
2. [Retrieving the Publications and Composing Links](#)
3. [Creating the FEM Rep and Retrieving its Rep Manager](#)
4. [Creating a Mesh on a Part](#)

5. [Creating a Property](#)
6. [Initializing and Creating the Design Exploration PLM Object](#)
7. [Retrieving the Scenario Manager](#)
8. [Creating the Structural Analysis Case with the FEM Rep](#)
9. [Creating and Initializing the Static Step](#)
10. [Retrieving the Structural Analysis Case Features Set](#)
11. [Creating Structural Analysis Case features](#)
12. [Creating the Bead Design Improvement Study with the Structural Analysis Case](#)
13. [Retrieving the Design Improvement Study features](#)
14. [Creating the Design Improvement Study features](#)
15. [Launching the Bead Design Improvement Study](#)

1. Retrieving the active Product

As a first step, the UC retrieves the active product.

```
...
Dim myEditor As Editor
Set myEditor = CATIA.ActiveEditor

Dim myProdService As PLMProductService
Set myProdService = myEditor.GetService("PLMProductService")

Dim myEntities As PLMEntities
Set myEntities = myProdService.EditedContent

Dim myRootProduct As VPMReference
Set myRootProduct = myEntities.Item(1)
...
```

2. Retrieving the Publications and Composing Links

In this step UC retrieves the publications aggregated by the root product and composes links.

```
...
Dim myPublications As VPMPublications
Set myPublications = myRootProduct.Publications

Dim myDesignSpaceSupport As VPMPublication
Set myDesignSpaceSupport = myPublications.GetItem("Design_Space")

' Composing link to the publication

Dim myRootOccurrence As VPMRootOccurrence
Set myRootOccurrence = myProdService.RootOccurrence

Dim myRepInstance As VPMRepInstance
Set myRepInstance = Nothing ' Because we use publications

Dim myLinkToDisconnectSupport As AnyObject
Set myLinkToDisconnectSupport = myProdService.ComposeLink(myRootOccurrence, myRepInstance, myDesignSpaceSupport)

...
```

3. Creating the FEM Rep and Retrieving its Rep Manager

In this step UC creates a finite element representation model on the product which was just opened.

For further information about creating the FEM Rep and retrieving its Rep Manager, refer to the article [Creating FEM representation](#).

4. Creating a Mesh on a Part

In this step UC retrieves the nodes and elements set from the FEM Representation, creates a mesh on a part and sets the mesh attributes.

For further information about the creation and management of mesh parts, refer to the article [Creating Mesh Part](#).

5. Creating a Property

In this step UC creates a solid section on the part and sets its individual attributes.

For further information about the creation and management of properties, refer to the article [Managing FEM section features](#).

6. Initializing and Creating the Design Exploration Simulation PLM Object

In this step UC creates a simulation object.

```
...
' Initializing the simulation PLM Object

Dim myInitService As SimInitializationService
Set myInitService = CATIA.GetSessionService("SimInitializationService")

myInitService.SetSimulationInitialization "SimDesignexplorationCreation", "SimOptimization"

' Creating the simulation PLM Object

Dim mySimPLMService As SIMPLMService
Set mySimPLMService = CATIA.GetSessionService("SIMPLMService")

Dim mySimulationReference As SimulationReference
mySimPLMService.PLCreate "NewDesignExplorationObject", "SMAfeaPLMNewSimu", myModel, mySimulationReference, myEditor
...
```

For further information about the initialization of a simulation object, refer to the article [Initializing a Simulation Object](#).

7. Retrieving the Scenario Manager

In this step UC retrieves the scenario manager.

```
...
Dim myScenarioManager As SimScenarioManager
Set myScenarioManager = mySimulationReference.GetItem("SimScenarioManager")
...
```

8. Creating the Structural Analysis Case with the FEM Rep

In this step UC creates a structural analysis case and associates it with the FEM rep.

For further information about the creation and management of structural analysis case, refer to the article [Creating and Executing a Simulation](#).

9. Creating and Initializing the Static Step

In this step UC creates a static step under the structural analysis case and sets its individual attributes.

For further information about the creation and management of steps, refer to the article [Creating and Executing a Simulation](#).

10. Retrieving the Structural Analysis Case Features Set

In this step UC retrieves the features set for the structural analysis case.

For further information about the management of structural analysis case features set, refer to the article [Creating and Executing a Simulation](#).

11. Creating Structural Analysis Case features

In this step UC creates and initializes features for the structural analysis case.

For further information about the creation and management of structural analysis features, refer to the article [Creating and Executing a Simulation](#).

12. Creating the Bead Design Improvement Study with the Structural Analysis case

In this step UC creates a Bead Design Improvement Study and associates it with the structural analysis case.

```
...
Dim myAnalysisCases As SimAnalysisCases
Set myAnalysisCases = myScenarioManager.AnalysisCases

Dim myBeadStudy As SimBeadDesignImprovementStudy
Set myBeadStudy= myAnalysisCases.AddAnalysisCase("SimBeadDesignImprovementStudy")

myBeadStudy.AnalysisCase = myAnalysisCase
...
```

13. Retrieving the Design Improvement Study Features Set

In this step UC retrieves the design exploration features set.

```
...
Dim myBeadFeatures As SimDesignImprovementFeatures
Set myBeadFeatures = myBeadFeatures.Features
...
```

14. Creating Design Improvement Study Features

In this step UC creates features for the simulation.

```
...
Dim myDesignSpace As SimDesignSpace
Set myDesignSpace = myFeatures.Add("SimDesignSpace")

' Setting feature support

Dim myLinkAccess As SimLinkAccess
Set myLinkAccess = myDesignSpace .GetItem("SimLinkAccess")
myLinkAccess.AddLink "MainSupport", myLinkToDisconnectSupport
...
```

You can find the available late types for features creation in the [Creation Late Types for Design Exploration Scenario Features](#) article.

15. Launching the Bead Design Improvement Study

In this step UC launch the topology design improvement study and shows the results.

```
...
Dim myExecutionService As SimExecutionService
Set myExecutionService = myEditor.GetService("SimExecutionService")

myExecutionService.BasicExecuteAll mySimulationReference

' switching to the results visualization workbench
CATIA.StartWorkbench "SMAHvcResultsVisualizationWorkbench"
...
```

Initializing and Executing a Bead Design Improvement Study

This article explains how to initialize and execute a Bead design improvement study.

Before you begin:

- Launch CATIA and then import and open the CAAScdfeaClampModel.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaDesignExplorationScenario\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- Make sure that you have all of the licenses required for launching execution and perform the solve operation in the interactive session for a similar simulation to test the licenses.
- Note: Creation of new PLM objects via scripting, as in this use case, requires an enabling license in addition to the interactive Roles needed to access the functionality.

Where to find the macro: [CAAScdfeaBeadDesignImprovementStudyWithPythonSource.htm](#)

This use case can be divided in 12 steps:

1. [Retrieving the active Product](#)
2. [Retrieving the Publications and Composing Links](#)
3. [Creating the FEM Rep and Retrieving its Rep Manager](#)
4. [Creating a Mesh on a Part](#)
5. [Creating a Property](#)
6. [Initializing and Creating the Design Exploration PLM Object](#)
7. [Retrieving the Scenario Manager](#)
8. [Creating the Structural Analysis Case with the FEM Rep](#)
9. [Creating and Initializing the Static Step](#)
10. [Retrieving the Structural Analysis Case Features Set](#)
11. [Creating Structural Analysis Case features](#)
12. [Creating the Bead Design Improvement Study with the Structural Analysis Case](#)
13. [Retrieving the Design Improvement Study features](#)
14. [Creating the Design Improvement Study features](#)
15. [Launching the Bead design improvement study](#)

1. Retrieves the active product

As a first step, the UC retrieves the active product

```
...
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
myEntities = myProdService.EditedContent
myEntity = myEntities.Item(1)
...
```

2. Retrieves the Publications and Composes Links

In this step UC retrieves the publications aggregated by the root product and composes links.

```
...
myPublications = myModel.Publications
myDesignSpaceSupport = myPublications.GetItem("Design_Space")

# Composing links
myLinkToDisconnectSupport = myProdService.ComposeLink(None, None, myDesignSpaceSupport)
...
```

As we use publication, we just need to give it as third argument to compose the link

3. Creates the FEM Rep and Retrieving its Rep Manager

In this step UC creates a finite element representation model on the product which was just opened.

For further information about creating the FEM Rep and retrieving its Rep Manager, refer to the article [Creating FEM representation](#).

4. Creates a Mesh on a Part

In this step UC retrieves the nodes and elements set from the FEM Representation, creates a mesh on a part and sets the mesh attributes.

For further information about the creation and management of mesh parts, refer to the article [Creating Mesh Part](#).

5. Creates a Property

In this step UC creates a solid section on the part and sets its individual attributes.

For further information about the creation and management of properties, refer to the article [Managing FEM section features](#).

6. Initializing and Creating the Design Exploration Simulation PLM Object

In this step UC creates a simulation object.

```
...
# Creating and Initializing the design exploration PLM Object
myInitService = CATIA.GetService("SimInitializationService")
myInitService.SetSimulationInitialization "SimDesignexplorationCreation", "SimOptimization"
mySimPLMService = CATIA.GetService("SIMPLMService")
mySimPLMService.PLMCreat "NewDesignExplorationObject", "SMAfeaPLMNewSimu", myModel, mySimulationReference, myEditor
...
```

For further information about the initialization of a simulation object, refer to the article [Initializing a Simulation Object](#).

7. Retrieves the Scenario Manager

In this step UC retrieves the scenario manager.

```
...
myScenarioManager = MySimulationRoot.GetItem("SimScenarioManager")
```

...

8. Creating the Structural Analysis Case with the FEM Rep

In this step UC retrieves the structural analysis case and associates it with the FEM rep.

For further information about the creation and management of structural analysis case, refer to the article [Creating and Executing a Simulation \(Python\)](#).

9. Creating and Initializing the Static Step

In this step UC creates a static step under the structural analysis case and sets its individual attributes.

For further information about the creation and management of steps, refer to the article [Creating and Executing a Simulation \(Python\)](#).

10. Retrieving the Structural Analysis Case Features Set

In this step UC retrieves the features set for the structural analysis case.

For further information about the management of structural analysis case features set, refer to the article [Creating and Executing a Simulation \(Python\)](#).

11. Creating Structural Analysis Case features

In this step UC creates and initializes features for the structural analysis case.

For further information about the creation and management of structural analysis features, refer to the article [Creating and Executing a Simulation \(Python\)](#).

12. Associates the Bead Design Improvement Study with the Structural Analysis Case

In this step UC retrieves the Bead design improvement study and associates it with the structural analysis case

```
...
myAnalysisCases = myScenarioManager.AnalysisCases
myBeadStudy = myAnalysisCases.AddAnalysisCase("SimBeadDesignImprovementStudy")
myBeadStudy.AnalysisCase = myAnalysisCase
...
```

13. Retrieving the Design Improvement Study Features Set

In this step UC retrieves the features set.

```
...
myBeadFeatures = myBeadStudy.Features
...
```

14. Creating Design Improvement Study Features

In this step UC creates features for the Bead design improvement study.

```
...
# Creating Design Space feature and setting its support
myDesignSpace = myFeatures.Add("SimDesignSpace")
myLinkAccess = myDesignSpace.GetItem("SimLinkAccess")
myLinkAccess.AddLink "MainSupport", myLinkToDisconnectSupport
...
```

You can find the available late types for features creation in the [Creation Late Types for Design Exploration Scenario Features](#) article.

15. Launches the Bead Design Improvement Study

In this step UC launch the Bead design improvement study and shows the results.

```
...
myExecutionService = myEditor.GetService("SimExecutionService")
myExecutionService.BasicExecuteAll (mySimulationReference)
...
```

Creation Late Types for Design Exploration Scenario Features

Technical Article

Abstract

The following reference article documents the different late types that can be used to create Design Exploration Scenario features.

Related Topics

[SimDesignExplorationFeatures Collection](#)

- [Constraints Late Types](#)
- [Design Variable Late Types](#)
- [Manufacturing Controls Features Late Types](#)
- [Objective Late Types](#)
- [Response Variable Late Types](#)
- [Shape ControlsFeatures Late Types](#)

Constraint Late Types

Constraints creation is managed by the **SimDesignExplorationFeatures** collection by calling the `Add` method from it.

A "late type" must be given to indicate which type of constraint is to be created:

Constraints	Late type
Center Of Gravity Constraint	SimCenterOfGravityConstraint
Displacement Constraint	SimDisplacementsConstraint
Fastener Force Constraint	SimFastenerForceConstraint
Frequency Constraint	SimFrequencyConstraint
General Constraint	SimGeneralConstraint
Mass Constraint	SimMassConstraint
Stress Constraint	SimStressConstraint
Reaction Force Constraint	SimReactionForceConstraint

Design Variable Features Late Types

Design Variable features creation is managed by the **SimDesignExplorationFeatures** collection by calling the `Add` method from it.

A "late type" must be given to indicate which type of Setup feature is to be created:

Design Variable feature	Late type
Design Space	SimDesignSpace

Design Space Sub-Features Late Types

Design space sub-features creation is managed by the **SimDesignExplorationFeatures** collection by calling the `AddSubFeature` method together with the parent **SimDesignSpace** or **SimLocalOptimizationArea** from it.

A "late type" must be given to indicate which type of Setup feature is to be created:

Design space sub-feature	Late type
Local Optimization Area	SimLocalOptimizationArea
In-Plane Control	SimInPlaneControl
Transition Control	SimTransitionControl
Preserved Region	SimPreservedRegion

Manufacturing Control Features Late Types

Manufacturing Controls features creation is managed by the **SimDesignExplorationFeatures** collection by calling the `Add` method from it.

A "late type" must be given to indicate which type of Manufacturing Controls feature is to be created:

Manufacturing Control feature	Late type
ALM Overhang Control	SimOverhangControl
Casting Control	SimCastingControl
Extrusion Control	SimExtrusionControl
Milling Control	SimMillingControl
Rib Control	SimRibControl

Response Variable Features Late Types

Response variable features creation is managed by the **SimDesignExplorationFeatures** collection by calling the `Add` method from it.

A "late type" must be given to indicate which type of load is to be created:

Response variable	Late type
Absolute Mass Response Variable	SimAbsoluteMassResponseVariable
Calculated Response Variable	SimCalculatedResponseVariable
Center Of Gravity Response Variable	SimCenterOfGravityResponseVariable
Displacement Response Variable	SimDisplacementResponseVariable
Fastener Force Response Variable	SimFastenerForceResponseVariable
Frequency Response Variable	SimFrequencyResponseVariable
Reaction Moment Response Variable	SimReactionMomentResponseVariable
Relative Mass Response Variable	SimRelativeMassResponseVariable
Moment Of Inertia Response Variable	SimMomentOfInertiaResponseVariable
Plastic Strain Magnitude Response Variable	SimPlasticStrainResponseVariable
Rotation Response Variable	SimRotationResponseVariable
Stiffness Response Variable	SimStiffnessResponseVariable
Stress Response Variable	SimStressResponseVariable

Objective features late types

Restraint features creation is managed by the **SimDesignExplorationFeatures** collection by calling the `Add` method from it.

A "late type" must be given to indicate which type of restraint is to be created:

Objective	Late type
Objective	SimNonParametricObjective

Shape Control features late types

Shape Control features creation is managed by the **SimDesignExplorationFeatures** collection by calling the `Add` method from it.

A "late type" must be given to indicate which type of shape control is to be created:

Shape Control	Late type
Bead Width Control	SimBeadWidthControl
Cyclic Symmetry Control	SimCyclicSymmetryControl
Penetration Check	SimPenetrationCheck
Symmetry Control	SimSymmetryControl
Thickness Control	SimThicknessControl

History

Version: 1 [February 2023] Document created

Initializing a Simulation Object

Technical Article

Abstract

The following reference section documents the required licensing for initializing various types of simulations.

- [Initializing a Simulation Object](#)

Initializing a Simulation Object

The first step in creating a new simulation is to call the `SetSimulationInitialization` method of **SimInitializationService**.

```
...
' Initializing the simulation PLM Object
Dim myInitService As SimInitializationService
Set myInitService = CATIA.GetSessionService("SimInitializationService")
myInitService.SetSimulationInitialization "SimDesignExplorationCreation", "SimOptimization"
...
```

The argument strings into the `SetSimulationInitialization` method maps to the App and Analysis Case of the Simulation as per the table below.

App	AppName Argument	Analysis Case	TypeName Argument	Recommended Roles
SIMULIA Mechanical Scenario Creation	SimMechanicalScenarioCreation	General	SimGeneral	SYE - Structural Analysis Engineer
		Structural	SimStructural	SSU - Structural Mechanics Engineer
		Thermal	SimThermal	
		Thermal	SimThermalStructural	
		Structural		
SIMULIA Structural Scenario Creation	SimStructuralScenarioCreation	Structural	SimStructural	SFO - Structural Performance Engineer
		Thermal	SimThermal	
		Thermal	SimThermalStructural	
		Structural		
SIMULIA Linear Structural Scenario Creation	SimLinearStructuralScenarioCreation	Structural	SimStructural	SLL - Structural Engineer
		Thermal	SimThermal	
SIMULIA Linear Structural Validation	SimLinearStructuralValidation	Structural	SimStructural	SRD - Structural Designer
		Thermal	SimThermal	
		Frequency	SimFrequency	
		Buckle	SimBuckle	
Design Exploration	SimDesignExplorationCreation	Optimization	SimOptimization	OPZ - Structural Generative Engineer

Once the `SetSimulationInitialization` has been called, a new simulation can be created by calling `PLMCreate` method of **SIMPLMService**.

```
...
' Creating the simulation PLM Object
Dim mySimPLMService As SIMPLMService
Set mySimPLMService = CATIA.GetSessionService("SIMPLMService")

Dim mySimulationReference As SimulationReference
mySimPLMService.PLMCreate "NewSimObject", "SMAfeaPLMNewSimu", myModel, mySimulationReference, myEditor
...
```

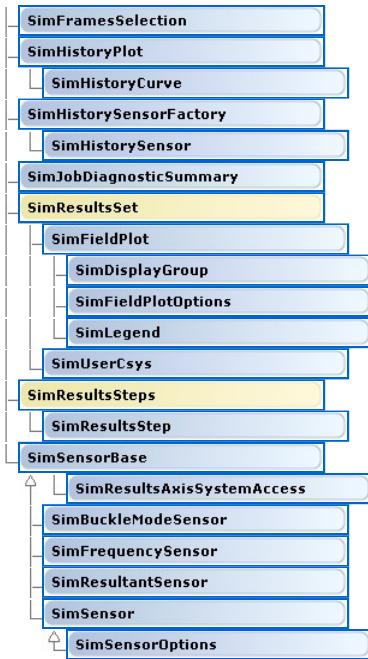
History

Version: 2 [Mar 2023] Document updated to remove deprecated method

Physics Results Explorer Object Model Map

See Also [Legend](#)





Use the **GetItem** method of the **SimResultRepManager** object to retrieve the **SimResultsManager** object.

```

Dim oSimResultRepManager As SimResultRepManager
...
Dim oSimResultsManager As SimResultsManager
Set oSimResultsManager = oSimResultRepManager.GetItem("SimResultsManager")
  
```

Opening a Simulation Document

This article explains how to open the .3dxml or SIM file and retrieve the results analysis case. This can be used to create and manage the results features.

Before you begin: Note that:

- **Opening .3dxml file:** You should first launch CATIA and then import the CAAScdfeaResults.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- **Opening SIM file:** You should first launch CATIA. Then open the Physics Results app. Rename the CAAScdfeaSimResults.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed to `CAAScdfeaSimResults.SMASim`. Then import this file.

This use case can be divided in two steps:

1. [Retrieving the Sim Results Manager](#)
2. [Retrieving the Results Analysis Case](#)

1. Retrieving the Sim Results Manager

As a first step, the below code retrieves editor, the simulation reference, the simulation results rep and the sim results manager.

```

...
Dim oEditor As Editor
Set oEditor = CATIA.ActiveEditor

Dim oSimulationReference As SimulationReference

Dim oProdService As PLMProductService
Set oProdService = oEditor.GetService("PLMProductService")

Dim oEntities As PLMEntities

If Not oProdService.EditedContent Is Nothing Then
  Set oEntities = oProdService.EditedContent

  Dim oEntity As PLMEntity
  Dim oSimObjEntity As PLMEntity

  For Each oEntity In oEntities
    If oEntity.GetCustomType = "SIMObjSimulationObjectGenericDS" Or oEntity.GetCustomType = "DesignSight" Then
      Set oSimObjEntity = oEntity
    End If
  Next

  Set oSimulationReference = oSimObjEntity

  Dim oSimResults As SimulationResults
  Set oSimResults = oSimulationReference.Results

  Dim oSimRepresentation As SimulationRepresentation
  Set oSimRepresentation = oSimResults.Item(1)

  Dim oSimRepManager As SimResultRepManager
  Set oSimRepManager = oSimRepresentation.Root
  
```

```

Dim oResultsManager As SimResultsManager
Set oResultsManager = oSimRepManager.GetItem("SimResultsManager")

End If
...

```

2. Retrieving the Results Analysis Case

As a first step, the below code retrieves the analysis case from results manager.

```

...
Dim oResultsAnalysisCases As SimResultsAnalysisCases
Set oResultsAnalysisCases = oResultsManager.ResultsAnalysisCases

Dim oResultsAnalysisCase As SimResultsAnalysisCase
Set oResultsAnalysisCase = oResultsAnalysisCases.Item(1)

...

```

Opening a Simulation Document

This article explains how to open the .3dxml or SIM file and retrieve the results analysis case. This can be used to create and manage the results features.

Before you begin: Note that:

- **Opening .3dxml file:** You should first launch CATIA and then import the CAAScdfeaResults.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- **Opening SIM file:** You should first launch CATIA. Then open the Physics Results app. Rename the CAAScdfeaSimResults.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed to `CAAScdfeaSimResults.SMASim`. Then import this file.

This use case can be divided in two steps:

1. [Retrieving the Sim Results Manager](#)
2. [Retrieving the Results Analysis Case](#)

1. Retrieving the Sim Results Manager

As a first step, the below code retrieves editor, the simulation reference, the simulation results rep and the sim results manager.

```

...
import sys, os
from com3dx import *
CATIA = get3dxClient()
ENUM = win32com.client.constants

oEditor = CATIA.ActiveEditor

oProdService = oEditor.GetService("PLMProductService")

oEntities = oProdService.EditedContent

oEntity = oEntities.Item(1)

oSimulationRoot = oEntity

oResults = oSimulationRoot.Results

oSimulationRepRes = oResults.Item(1)

oRootRepRes = oSimulationRepRes.Root

oResultsManager = oRootRepRes.GetItem("SimResultsManager")

...

```

2. Retrieving the Results Analysis Case

As a first step, the below code retrieves the analysis case from results manager.

```

...
oResultsAnalysisCases = oResultsManager.ResultsAnalysisCases

oResultsAnalysisCase = oResultsAnalysisCases.Item(1)

...

```

Retrieving the field and history variables

This article explains how to get the field and history variable names specified in the output request. It also shows how to retrieve the region and sub regions available for the history output

Where to find the macro: [CAAScdfeaResultsGetOutputVariablesSource.htm](#)

This use case can be divided in 5 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Retrieves the step](#)
3. [Retrieves the field variables from step](#)
4. [Retrieves the history variables from step](#)
5. [Retrieves the region and sub region for history variables](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case. You can use `VB_HistoryPlot.SMASim` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API

CD-ROM is installed.

On opening the model,Select yes option from the dialogue to activate the results.

[Retrieve the analysis case](#)

2. **Retrieves the step**

This step shows how to get the step from the case.

```
...
Dim oResultsSteps
Set oResultsSteps = oResultsAnalysisCase.ResultsSteps

Dim oResultsStep
Set oResultsStep = oResultsSteps.Item(1)
...
```

3. **Retrieves the field variables from step**

This step shows how to get the field variables from the step. The list of avialable field names will be returned. These variable can then be used to create Plots, Sensors etc.

```
...
Dim idFrame As Double
idFrame = 0
Dim ieSource As SimResultsSource
ieSource = SimFieldSourceResults

The available options for SimResultsSource are:
SimFieldSourceResults,
SimFieldSourceDiagnostic,
SimFieldSourcePreLoad,
SimFieldSourceHistory

Dim oFieldVariables()
oResultsStep.GetAvailableFields idFrame, ieSource, oFieldVariables
...
```

4. **Retrieves the history variables from step**

This step shows how to get the history variables from the step. The list of avialable history names will be returned. These variable can then be used to create history plots, history sensors.

```
...
Dim idFrame As Double
idFrame = 0
Dim ieSource As SimResultsSource
ieSource = SimFieldSourceHistory

Dim oHistoryVariables()
oResultsStep.GetAvailableFields idFrame, ieSource, oHistoryVariables
...
```

5. **Retrieves the region and sub region for history variables**

This step shows how to get the history region and sub region informationfor the given variable. These can be used to specify support while creating the history plot.

```
...
Dim iVariableString As String
oVariableString = "U"

Dim oRegionNames() // All the region names
Dim oSubRegionIndex() // Number of sub regions per region
Dim oSubRegionNames()// Global list of sub regions
oResultsStep.GetHistoryRegionsAndSubRegions iVariableString, oRegionNames, oSubRegionIndex, oSubRegionNames
...
```

Retrieving the field and history variables

This article explains how to get the field and history variable names specified in the output request. It also shows how to retrieve the region and sub regions avaialble for the history output

Where to find the macro: [CAAScdfeaResultsGetOutputVariablesSourcePython.htm](#)

This use case can be divided in 5 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Retrieves the step](#)
3. [Retrieves the field variables from step](#)
4. [Retrieves the history variables from step](#)
5. [Retrieves the region and sub region for history variables](#)

1. **Opens the document and retrieves the results analysis case**

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case. You can use `VB_HistoryPlot.SMASim` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

On opening the model,Select yes option from the dialogue to activate the results.

[Retrieve the analysis case](#)

2. **Retrieves the step**

This step shows how to get the step from the case.

```
...
oResultsSteps = oResultsAnalysisCase.ResultsSteps
oResultsStep = oResultsSteps.Item(1)
...
```

3. Retrieves the field variables from step

This step shows how to get the field variables from the step. The list of available field names will be returned. These variable can then be used to create Plots, Sensors etc.

```
...
idFrame = 0
ieSource = ENUM.SimFieldSourceResults

The available options for SimResultsSource are:
SimFieldSourceResults,
SimFieldSourceDiagnostic,
SimFieldSourcePreLoad,
SimFieldSourceHistory

oFieldVariables = oResultsStep.GetAvailableFields (idFrame, ieSource, ())
...
```

4. Retrieves the history variables from step

This step shows how to get the history variables from the step. The list of available history names will be returned. These variable can then be used to create history plots, history sensors.

```
...
idFrame = 0
ieSource = ENUM.SimFieldSourceHistory
oHistoryVariables = oResultsStep.GetAvailableFields (idFrame, ieSource, ())
...
```

5. Retrieves the region and sub region for history variables

This step shows how to get the history region and sub region information for the given variable. These can be used to specify support while creating the history plot.

```
...
iVaribleString = "U"
oRegionNames, oSubRegionIndex, oSubRegionNames = oResultsStep.GetHistoryRegionsAndSubRegions (iVaribleString, (), (), ())
...
```

Creating a field plot

This article explains how to retrieve the results features.

This use case can be divided in 9 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Retrieves the features from the results set](#)
3. [Identifiers to retrieve the features](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case.

[Retrieve the analysis case](#)

2. Retrieves the features from the results set

In this step we retrieve the results set and then get the features from it.

```
Dim oResultsSet As SimResultsSet
Set oResultsSet = oResultsAnalysisCase.GetSet("FieldPlotSet")

'We can get the feature either by its index or by its name from the results set

'The following step shows how to retrieve the plot by its index
Dim oFieldPlot As SimFieldPlot
Set oFieldPlot = oResultsSet.Item(1)

'The following step shows how to retrieve the plot by its name
Dim oFieldPlot1 As SimFieldPlot
Set oFieldPlot1 = oResultsSet.Item("Plot U.1")
```

3. Identifiers to retrieve the features

Following is the list of string by which the feature sets can be retrieved:

1. "FieldPlotSet"
2. "SensorSet"
3. "ResultantSensorSet"
4. "FrequencySensorSet"
5. "BuckleModeSensorSet"
6. "HistoryPlotSet"
7. "UserCsysSet"
8. "EnvelopePlotSet"
9. "LLCCombinationPlotSet"

```
10."DisplayGroupSet"
```

Creating a field plot

This article explains how to open the results and create a field plot.

Where to find the macro: [CAAScdfeaResultsFieldPlotSource.htm](#)

This use case can be divided in 9 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Retrieves the plot from the results set](#)
3. [Gets a list of creatable plot Ids from the plot definition](#)
4. [Creates the field plot through template](#)
5. [Sets the display group to the plot](#)
6. [Sets the deformation scale factor](#)
7. [Retrieves the legend from the plot](#)
8. [Retrieves the plot type](#)
9. [Retrieves the step and frame from the plot](#)
10. [Deletes the plot](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case.

[Retrieve the analysis case](#)

2. Retrieves the plot from the results set

In this step we retrieve the results set and then get the first plot from the results set.

By default the Undeformed plot is the first plot.

```
...
Dim oFieldPlots As SimResultsSet
Set oFieldPlots = oResultsAnalysisCase.GetSet("FieldPlot")

Dim oFieldPlot As SimFieldPlot
Set oFieldPlot = oFieldPlots.Item(1)
...
```

3. Gets a list of creatable plot Ids from the plot definition

This step shows how to get the creatable plot ids from the plot.

If ilFields is an empty list, the list of existing fields in the model is used.

The returned list of Field Plot Ids is filtered to exclude the Ids of Field Plots referencing fields that are not in the list or fields that are not in the model.

If ibIncludeDefaults is True, include the internal Field Plot definitions in addition to definitions passed to @href #AddFieldPlotDefinitions.

If ibIncludeUndMesh is True, include the Field Plot Id for the undeformed mesh.

sId is the returned list of Field Plot Id strings.

```
...
Dim ilFields() As Variant
Dim ibIncludeDefaults As Boolean
Dim ibIncludeUndMesh As Boolean
Dim sId() As Variant
oResultsAnalysisCase.GetListOfFieldPlotIDs ilFields, ibIncludeDefaults, ibIncludeUndMesh, sId
...
```

4. Creates the field plot through template

This step shows how to create a field plot from the results analysis case retrieved from the earlier step.

```
...
Dim sId As String
sId = "Contour_Stress_Tresca"
Dim oPlot As SimFieldPlot
Set oPlot = oResultsAnalysisCase.CreateFieldPlotByID(sId, True)
...
```

5. Sets the display group to the plot

The following code sets the already created display groups to the plot.

```
...
'First we retrieve the display group from the results set
Dim oDisplayGroupSet As SimResultsSet
Set oDisplayGroupSet = oResultsAnalysisCase.GetSet("DisplayGroupSet")
Dim oDisplayGroup As SimDisplayGroup
Set oDisplayGroup = oDisplayGroupSet.Item(1)

Dim MyDispatch As CATBaseDispatch
Set MyDispatch = oDisplayGroup
Dim MySupports(0)
Set MySupports(0) = MyDispatch
oPlot.DisplayGroup = MyDispatch
oPlot.Refresh
...
```

6. Sets the deformation scale factor

The following code sets the deformation scale factor to the plot.

```
...
oPlot.SetDeformationScaleFactors 2, 2, 2
...
```

7. Retrieves the legend from the plot

Here we retrieve the plot legend.

```
...
Dim oPlotLegend As SimLegend
Set oPlotLegend = oPlot.Legend
...
```

8. Retrieves the plot type

More options for setting SimFieldPlotTypes can be found at [FieldPlotEnums](#).

```
...
Dim oPlotType As SimFieldPlotTypes
Set oPlotType = oPlot.Type
...
```

9. Retrieves the step and frame from the plot

Following code retrieves the step and frame information from the plot.

```
...
Dim nbFrame As Long
Set nbFrame = oPlot.GetCurrentFrame True

Dim sStep As String
oPlot.GetStepAndFrame sStep, nbframe, 0, False
...
```

10. Deletes the plot

Deletes the plot feature.

```
...
oResultsAnalysisCase.DeleteFeature oPlot
```

Managing field plot options

This article explains how to open the results, retrieve the field plot and modify the rendering options.

Where to find the macro: [CAAScdFeaFieldPlotOptionsSource.htm](#)

This use case can be divided in 5 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Retrieves the existing field plot](#)
3. [Gets the field plot options from the field plot](#)
4. [Sets the field plot options](#)
5. [Updates the field plot](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case. You can use `CAAScdFeaResults.3dxml` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdFeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

On opening the model, select "Results" in the action bar to go to Physics Results Explorer.

[Retrieve the analysis case](#)

2. Retrieves the existing field plot

This step shows how to get the field plot from the results analysis case retrieved from the earlier step.

```
...
Dim oFieldPlots
Set oFieldPlots = oResultsAnalysisCase.GetSet("FieldPlots")

Dim oFieldPlot As SimFieldPlot
Set oFieldPlot = oFieldPlots.Item(2)
...
```

3. Gets the field plot options from the field plot

This step shows how to get the field plot options from the field plot.

```
...
Dim oFieldOptions As SimFieldPlotOptions
Set oFieldOptions = oFieldPlot.GetItem("SimFieldPlotOptions")
...
```

4. Sets the field plot options

The following code sets the field plot options

```
...
oFieldOptions.SetShowLabels True, True
```

```
More options for setting SimThresholdType and SimVisibleEdges can be found at FieldPlotEnums.
Dim eThreshold As SimThresholdType
eThreshold = SimThresholdUpperLimit
oFieldOptions.SetThreshold eThreshold ,15, TRUE
oFieldOptions.Transparency = True

Dim eVisibleEdges As SimVisibleEdges
eVisibleEdges = SimOutline
oFieldOptions.SetVisibleEdges SimOutline, 25
...
```

5. Updates the field plot

Following code updates the field plot with new attributes.

```
...
oFieldOptions.Update
...
```

Retrieving and managing plot legend

This article explains how to retrieve and manage the field plot legend.

Where to find the macro: [CAAScdfeaResultsPlotLegendSource.htm](#)

This use case can be divided in 9 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Creates the field plot through template](#)
3. [Retrieves the legend from the plot](#)
4. [Gets and sets the minimum and maximum value for the legend](#)
5. [Gets and Sets the color modifiers](#)
6. [Gets and Sets the legend style](#)
7. [Gets and sets the number of legend colors](#)
8. [Sets the number format](#)
9. [Specifies the animation range](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case.

[Retrieve the analysis case](#)

2. Creates the field plot through template

This step shows how to create a field plot from the results analysis case retrieved from the earlier step.

```
...
Dim sId As String
sId = "Contour_Stress_Tresca"
Dim oPlot As SimFieldPlot
Set oPlot = oResultsAnalysisCase.CreateFieldPlotByID(sId, True)
...
```

3. Retrieves the legend form the plot

Here we retrieve the plot legend.

```
...
Dim oPlotLegend As SimLegend
Set oPlotLegend = oPlot.Legend
...
```

4. Gets and sets the minimum and maximum value for the legend

In this step we retrieve the minimum and maximum values for the legend.

```
...
Dim nbMinvalue As double
Dim nbMaxvalue As double

oPlotLegend.SetMinMaxValue 100, 10000, True
oPlotLegend.GetMinMaxValue nbMinvalue, nbMaxvalue, True, False
...
```

5. Gets and Sets the color modifiers

Following code gets and sets the smooth and inverse colors options.

```
...
Dim bSmooth As boolean
Dim bInverse As boolean

oPlotLegend.GetColorModifiers bSmooth, bInverse
oPlotLegend.SetColorModifiers True, True
...
```

6. Gets and Sets the legend style

The legend styles can be either SMAMpaLegendStyle_Detailed or SMAMpaLegendStyle_Simplified.

```
...
Dim sStyle As String
sStyle = oPlotLegend.Style
oPlotLegend.Style = "SMAMpaLegendStyle_Detailed"
...
```

7. Gets and sets the number of legend colors

In the following code we get and set the number of colors for the legend.

```
...
Dim nbColors As Long
nbColors = oPlotLegend.NbColors
oPlotLegend.NbColors = nbColors * 2
...
```

8. Sets the number format

The below code sets the numerical format used for result values to scientific. The various options are

Automatic = 0, Scientific = 1, Decimal = 2

```
...
oPlotLegend.NumberFormat = 1
...
```

9. Specifies the animation range

Sets the legend animation range.

```
...
oPlotLegend.StartAnimationRange 200, 5000, True
...
```

Configuring multiviews and adding plots to it

This article explains how to configure multiviews and add plots to it.

Where to find the macro: [CAAScdfeaResultsMultiViewSource.htm](#)

This use case can be divided in 6 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Retrieves the multi view manager](#)
3. [Retrieves the plots list from the analysis case](#)
4. [Applies the given multi view configuration to current viewer](#)
5. [Adds a plot in multi view configuration at given position](#)
6. [Sets the synchronization mode for viewpoints](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case. You can use `VB_Composite_Analysis.SMASim` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Before proceeding further, create 3 plots and make undeformed plot as an active plot.

[Retrieve the analysis case](#)

2. Retrieves the multi view manager

This step shows how to retrieve the multi view manager.

```
...
Dim MultiViewManager As SimMultiView
Set MultiViewManager = oResultsManager.MultiViewManager
...
```

3. Retrieves the plots list from the analysis case

This step shows how to get list of plots from the analysis case.

```
...
Dim oFieldPlots As SimResultsSet
Set oFieldPlots = oResultsAnalysisCase.GetSet("FieldPlots")

Dim iPos As Integer
iPos = 1

Dim oFieldPlot As SimFieldPlot
Set oFieldPlot = oFieldPlots.Item(2)
...
```

4. Applies the given multi view configuration to current viewer

In this step we apply the multi view configuration to the current viewer.

The available configurations are:

- * SimVPFourSquare to apply four views (2x2).
- * SimVPTriLeft to apply four views with three views on left and one main view on right.
- * SimVPTriBottom to apply four views with three views on bottom and one main view on top.
- * SimVPTwoRow to apply two views top and bottom.
- * SimVPTwoColumn to apply two views left and right.
- * SimVPSingle to apply default single view.

```
...
MultiViewManager.Configuration = SimVPFourSquare
...
```

5. Adds a plot in multi view configuration at given position

Following code shows how to add plot in multi view configuration at given position.

iPos is the position at which the plot has to be added.

Following are the iPos values to be used for various configurations.

- * VPFourSquare 1-Top left, 2- Top right, 3-Bottom left, 4-Bottom right.
- * VPTriLeft 1- Main view on right, 2-Top left, 3-Left middle, 4- Bottom left.
- * VPTriBottom 1-Top main view, 2- Bottom left, 3-Bottom middle, 4- Bottom right.
- * VPTwoRow 1- Top, 2- Bottom.
- * VPTwoColumn 1- Left, 2-Right.

```
...
iPos = 2
Set oFieldPlot = oFieldPlots.Item(3)
MultiViewManager.AddPlot oFieldPlot, iPos

iPos = 4
Set oFieldPlot = oFieldPlots.Item(4)
MultiViewManager.AddPlot oFieldPlot, iPos
...
```

6. Sets the synchronization mode for viewpoints

Setting synchronization mode TRUE will propagate actions performed such as Pan, Zoom, Rotate in one viewpoint to other viewpoints.

Setting synchronization mode FALSE will allow actions performed such as Pan, Zoom, Rotate in only one viewpoint.

Following code sets the synchronization mode for viewpoints.

```
...
MultiViewManager.Synchronize = False
...
```

Creating a history plot

This article explains how to open the results and create a history plot.

Where to find the macro: [CAAScdFeaResultsHistoryPlotSource.htm](#)

This use case can be divided in 13 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Creates the history plot](#)
3. [Get the step Id ,name and step description](#)
4. [Sets the attributes to the history plot](#)
5. [Sets the support for the history plot](#)
6. [Sets step, frame and load case information](#)
7. [Sets the complex type and angle](#)
8. [Sets name to history plot](#)
9. [Updates the history plot](#)
10. [Returns the minimum and maximum values](#)
11. [Gets the list of curves present in the history plot](#)
12. [Exports the curve values in the specified file format](#)
13. [Returns the minimum and maximum values of the curve](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case. You can use `VB_HistoryPlot.SMASim` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdFeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

On opening the model, Select yes option from the dialogue to activate the results.

[Retrieve the analysis case](#)

2. Creates the history plot

This step shows how to create a history plot from the results analysis case retrieved from the earlier step.

```
...
Dim oHistoryPlot1
Set oHistoryPlot1 = oResultsAnalysisCase.CreateHistoryPlot
...
```

3. Get the step Id ,name and step description

This step shows how to get the step Id, name and step description.

```
...
dim oSteps As SimResultsSteps
Set oSteps = oResultsAnalysisCase.ResultsSteps
```

```
dim oResultsStep As SimResultsStep
Set oResultsStep = oSteps.Item(1)

dim stepId As String
dim stepName As String
dim stepDescription As String

oResultsStep.GetIdentifier stepId, stepName, stepDescription

...
```

4. Sets the attributes to the history plot

The following code sets the history plot attributes. In this step we set the history plot attributes. More options for history plot attributes can be found at [HistoryplotEnums](#).

```
...
oHistoryPlot1.Variable = "UT"
oHistoryPlot1.ComponentQuantity = SimVector_Component_1
...
```

5. Sets the support for the history plot.

The following code sets support for creating the history plot such as regions and sub-regions. If user wants to select the whole model, no need to call this method.

```
...
Dim MySupports(0)
MySupports(0) = "INP2SIM_NSET_Nodal_Probe2"

Dim MySupports1(0)
MySupports1(0) = 3

Dim MySupports2(2)
MySupports2(0) = "Node 4"
MySupports2(1) = "Node 36"
MySupports2(2) = "Node 87"

oHistoryPlot1.SetSupport MySupports, MySupports1, MySupports2
...
```

6. Sets the step, frame and load case information

Following code sets the step, frame and load case information.

stepId is the persistent ID of the step.

The Frame index starts from 1. This option is needed only in case of multiple load cases. In the following code we have used 0 as the frame index.

The load case index starts from 1. It is needed if a load case is present in the step. In the following code we have used 0 as the load case index.

```
...
oHistoryPlot1.SetStepAndFrame stepId, 0, 0
...
```

Using following code we can set all available time based steps.

```
...
oHistoryPlot1.SetAllTimeBasedSteps
...
```

7. Sets the complex type and angle

Following code sets the complex type and angle. The complex options are dependent on the complex options that are set while creating the step.

More options for setting SimComplexValues can be found at [HistoryplotEnums](#).

```
...
Dim ieComplexValue As SimComplexValues
ieComplexValue = SimEnvelopeMax

Dim iValueAtAngle As double
iValueAtAngle = 30

oHistoryPlot1.SetComplex ieComplexValue, iValueAtAngle
...
```

8. Sets name to history plot

Following code sets the name to history plot.

```
...
oHistoryPlot1.Name = "Test History Plot 1"
...
```

9. Updates the history plot

Following code creates and displays the history plot in the XY viewer. This method needs to be called at the end after setting the attributes.

```
...
oHistoryPlot1.Update
...
```

10. Returns the minimum and maximum values.

This step returns the minimum and maximum values. It considers all the curves and gives the absolute min and max value.

```
...
Dim dMin As Double
Dim dMax As Double
oHistoryPlot1.GetMinMaxValues dMin, dMax
```

11. Gets the list of curves present in the history plot

Following code returns the list of curves present in the history plot.

```
...
Dim olsHistoryCurves As CATSafeArrayVariant
oHistoryPlot1.GetHistoryCurveList olsHistoryCurves
...
```

12. Exports the curve values in the specified file format either to disc or in the database as a VPMDocument

(a) Saving the file on disc:

Following code exports the curve values in the specified file format at the given location.

icsFileName is the file name. If not specified, the name of the curve will be taken by default.

icsFileLocation is the The location at which the exported file needs to be created.

More options for setting ieFileType can be found at [HistoryplotEnums](#).

```
...
Dim icsFileName As String
icsFileName = "ExportedHistCurve"

Dim icsFileLocation As String
icsFileLocation="C:\temp"

Dim ieFileType As SimFileType
ieFileType = SimXlsx

Dim oHistoryCurve1 As SimHistoryCurve
Set oHistoryCurve1 = olsHistoryCurves.Item(1)

oHistoryCurve1.Export icsFileName,icsFileLocation,ieFileType
...
```

(b) Saving the file in data base as a VPMDocument:

Following code exports the curve values in the specified file format in the data base.

icsFileName is the file name. If not specified, the name of the curve will be taken by default.

More options for setting ieFileType can be found at [HistoryplotEnums](#).

```
...
Dim icsFileName As String
icsFileName = "ExportedHistCurve"

Dim ieFileType As SimFileType
ieFileType = SimCsv

Dim oHistoryCurve1 As SimHistoryCurve
Set oHistoryCurve1 = olsHistoryCurves.Item(1)

oHistoryCurve1.ExportToPLMDoc icsFileName,ieFileType
...

Note: The user will have save the document/session to see the exported file in the database.
```

13. Returns the minimum and maximum values of the curve

This step returns the minimum and maximum values of the curve.

```
...
Dim dMin As Double
Dim dMax As Double
oHistoryCurve1.GetMinMaxValues dMin, dMax
```

Creating a history plot

This article explains how to open the results and create a history plot.

Where to find the macro: [CAAScdfeaResultsHistoryPlotSourcePython.htm](#)

This use case can be divided in 13 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Creates the history plot](#)
3. [Get the step Id_name and step description](#)
4. [Sets the attributes to the history plot](#)
5. [Sets the support for the history plot](#)
6. [Sets step, frame and load case information](#)
7. [Sets the complex type and angle](#)
8. [Sets name to history plot](#)
9. [Updates the history plot](#)
10. [Returns the minimum and maximum values](#)
11. [Gets the list of curves present in the history plot](#)

12. [Exports the curve values in the specified file format](#)
13. [Returns the minimum and maximum values of the curve](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case. You can use `VB_HistoryPlot.SMASim` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

On opening the model, Select yes option from the dialogue to activate the results.

[Retrieve the analysis case](#)

2. Creates the history plot

This step shows how to create a history plot from the results analysis case retrieved from the earlier step.

```
...
oHistoryPlot1 = oResultsAnalysisCase.CreateHistoryPlot()
```

3. Get the step Id ,name and step description

This step shows how to get the step Id, name and step description.

```
...
oSteps = oResultsAnalysisCase.ResultsSteps
oResultsStep = oSteps.Item(1)
stepId, stepName, stepDescription = oResultsStep.GetIdentifier (), (), ()
```

4. Sets the attributes to the history plot

The following code sets the history plot attributes. In this step we set the history plot attributes. More options for history plot attributes can be found at [HistoryplotEnums](#)

```
...
oHistoryPlot1.Variable = ("UT")
oHistoryPlot1.ComponentQuantity = ENUM.SimVector_Component_1
```

5. Sets the support for the history plot.

The following code sets support for creating the history plot such as regions and sub-regions. If user wants to select the whole model, no need to call this method.

```
...
MySupports = ["INP2SIM_NSET_Nodal_Probe2"]
MySupports1 = [3]
MySupports2 = ["Node 4", "Node 36", "Node 87"]
oHistoryPlot1.SetSupport (MySupports, MySupports1, MySupports2)
```

6. Sets the step, frame and load case information

Following code sets the step, frame and load case information.

`stepId` is the persistent ID of the step.

The Frame index starts from 1. This option is needed only in case of multiple load cases. In the following code we have used 0 as the frame index.

The load case index starts from 1. It is needed if a load case is present in the step. In the following code we have used 0 as the load case index.

```
...
oHistoryPlot1.SetStepAndFrame (stepId, 0, 0)
```

Using following code we can set all available time based steps.

```
...
oHistoryPlot1.SetAllTimeBasedSteps()
```

7. Sets the complex type and angle

Following code sets the complex type and angle. The complex options are dependent on the complex options that are set while creating the step.

More options for setting `SimComplexValues` can be found at [HistoryplotEnums](#).

```
...
ieComplexValue = ENUM.SimEnvelopeMax
iValueAtAngle = 30
oHistoryPlot1.SetComplex (ieComplexValue, iValueAtAngle)
```

8. Sets name to history plot

Following code sets the name to history plot.

```
...
oHistoryPlot1.Name = ("Test History Plot 1")
...
```

9. Updates the history plot

Following code creates and displays the history plot in the XY viewer. This method needs to be called at the end after setting the attributes.

```
...
oHistoryPlot1.Update()
...
```

10. Returns the minimum and maximum values.

This step returns the minimum and maximum values. It considers all the curves and gives the absolute min and max value.

```
...
dMin, dMax = oHistoryPlot1.GetMinMaxValues (0, 0)
```

11. Gets the list of curves present in the history plot

Following code returns the list of curves present in the history plot.

```
...
oHistCurveList = oHistoryPlot1.GetHistoryCurveList(())
oHistCurve = oHistCurveList[0]
...
```

12. Exports the curve values in the specified file format either to disc or in the database as a VPMDocument

(a) Saving the file on disc:

Following code exports the curve values in the specified file format at the given location.

icsFileName is the file name. If not specified, the name of the curve will be taken by default.

icsFileLocation is the The location at which the exported file needs to be created.

More options for setting ieFileType can be found at [HistoryplotEnums](#).

```
...
iFileName = "History_Curve1"
iFileLocation="C:\\temp"
oHistCurve.Export (iFileName, iFileLocation, ENUM.SimXlsx)
```

(b) Saving the file in data base as a VPMDocument:

Following code exports the curve values in the specified file format in the data base.

icsFileName is the file name. If not specified, the name of the curve will be taken by default.

More options for setting ieFileType can be found at [HistoryplotEnums](#).

```
...
oHistCurve1 = oHistCurveList[1]
iFileName1 = "History_Curve_PLM"
oHistCurve1.ExportToPLMDoc (iFileName1, ENUM.SimCsv)
```

Note: The user will have save the document/session to see the exported file in the database.

13. Returns the minimum and maximum values of the curve

This step returns the minimum and maximum values of the curve.

```
...
dMin, dMax = oHistoryCurve1.GetMinMaxValues (0, 0 )
```

Creating a sensor

This article explains how to open the results and create a sensor.

Before you begin: Note that:

- **Opening .3dxml file:** You should first launch CATIA and then import the CAAScdfeaResultsMultiLoadCase.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: [CAAScdfeaResultsSensorSource.htm](#)

This use case can be divided in 11 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Creates the sensor](#)
3. [Sets the attributes to the sensor](#)
4. [Sets the axis to the sensor](#)
5. [Sets support to the sensor](#)
6. [Creates the frame selection](#)
7. [Sets the frame selection attributes](#)

8. [Sets the frame selection in the sensor](#)
9. [Sets the sensor parameters](#)
10. [Updates the sensor](#)
11. [Retrieves the sensor](#)
12. [Show annotation on the sensor](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case.

[Retrieve the analysis case](#)

2. Creates the sensor

This step shows how to create a sensor from the results analysis case retrieved from the earlier step.

```
...
Dim oLocalSensor As SimSensor
Set oLocalSensor = oResultsAnalysisCase.CreateSensor
...
```

Creates the sensor by Id

Creates a sensor using field plot definition defined in XML description library files. It takes input as the ID of the Field Plot as specified in the XML file as sensor if this method is used no need to set the sensor attributes mentioned in Step 3.

we can use the GetListOfFieldPlotIDs of ResultsAnalysisCase to get the list of plot ids. This plot ids can be used to create the sensor.

```
...
Dim csSensorID As String
Dim oLocalSensor As SimSensor
Set oLocalSensor = oResultsAnalysisCase.CreateSensorByID csSensorID
...
```

3. Sets the attributes to the sensor

The following code sets the sensor attributes.

```
...
oLocalSensor.VariableType = "S"
oLocalSensor.ProcessingType = SimQuantity
oLocalSensor.ComplexValue = SimValueAtAngle
oLocalSensor.Location = SimCentroids
oLocalSensor.ComplexValueAtAngle = 10
oLocalSensor.TransformType = SimUserModelAxis
oLocalSensor.QuantityType = "Mid Principal"
...
```

4. Sets the axis to the sensor

There are three ways in which we can set the axis as a transformation.

a) Using the axes available in the SIM: The exact name of the coordinate system needs to be specified.

```
...
oLocalSensor.TransformType = SimUserDefined
oLocalSensor.CoordinateSystem = "Orientation-Rib02--1-1"
...
```

b) The User Csys created in Physics results:

```
...
Dim oResultsCsysSet As SimResultsSet
Set oResultsCsysSet = oResultsAnalysisCase.GetSet("UserCsysSet")
Dim oUserCsys As SimUserCsys
Set oUserCsys = oResultsCsysSet.Item(1)

oLocalSensor.TransformType = SimUserDefined
Dim oAxisSystemAxis As SimResultsAxisSystemAccess
Set oAxisSystemAxis = oLocalSensor.GetItem("SimResultsAxisSystemAccess")

oAxisSystemAxis.Axis = oUserCsys
...
```

c) The model axes created at the product level: First the axis needs to be retrieved from the product, then set it to the sensor as below. Refer the sample script to see how to retrieve axis from the product.

```
...
oLocalSensor.TransformType = SimUserModelAxis
```

```
Set AxisSystems = MyPart.AxisSystems
Set AxisSystem = AxisSystems.Item(2)
```

```
Dim oAxisSystemAxis As SimResultsAxisSystemAccess
Set oAxisSystemAxis = oLocalSensor.GetItem("SimResultsAxisSystemAccess")
oAxisSystemAxis.Axis = AxisSystem
...
```

Note: For setting the axis call the "TransformType" and "QuantityType" in the same order as specified in Step 3. The TransformType

5. Sets support to the sensor

Here we set named display group as a support to the sensor.

```
...
```

```
'First we retrieve the display group from the results set
Dim oDisplayGroupSet As SimResultsSet
Set oDisplayGroupSet = oResultsAnalysisCase.GetSet("DisplayGroupSet")
```

```

Dim oDisplayGroup As SimDisplayGroup
Set oDisplayGroup = oDisplayGroupSet.Item(1)

Dim MyDispatch As CATBaseDispatch
Set MyDispatch = oDisplayGroup

Dim MySupports(0)
Set MySupports(0) = MyDispatch

LocalSensor.Support = MySupports

...

```

6. Creates the frame selection

Here we create a frame selection object.

```

...
Dim oFrameSelector As SimFramesSelection
Set oFrameSelector = oResultsAnalysisCase.CreateFrameSelector
...

```

7. Sets the frame selection attributes

In this step we set the frame selection attributes. More options for frames selection can be found at [Frames Selection](#)

...

a) Setting pre defined frame selection enumns in the sensor:

```
oFrameSelector.SymbolicFrameRange = SimAllFrames
```

b) Setting custom frame selection to the sensor:

1) Retrieving the Step ID and setting it to the frame selector
 Dim oResultsSteps As SimResultsSteps
 Set oResultsSteps = ResultsAnalysisCase.ResultsSteps

```
Dim oResultsStep As SimResultsStep
Set oResultsStep = oResultsSteps.Item(2)
```

```
Dim sID As String
Dim sStepName As String
Dim sStepName As String
Dim sDescription As String
oResultsStep.GetIdentifier sID, sStepName, sDescription
Dim lsStepID(0)
lsStepID(0) = sID
```

```
FrameSelector.StepList = lsStepID
```

2) Retrieving the Load case ID and setting it to the frame selector
 Note: It is mandatory to set the Load Case, if the selected step has one.

```
Dim oGlobalLoadCaseIDList()
Dim oGlobalLoadCaseIDNames()
Dim oLoadCaseIDListInStep()
```

The following method returns all the Load case ID avaialble in the case. The order of the list is the same as the user has created load case. One can get the indices of the Load case available in that step using GetLoadCases() method as shown below
 oResultsManager.GetLoadCaseIdentifier oGlobalLoadCaseIDList, oGlobalLoadCaseIDNames
 oResultsStep.GetLoadCases oLoadCaseIDListInStep
 Dim iLCIdentifierString As String
 Dim iLoadCaseIDListToSet(0)

```
iLCIdentifierString = oGlobalLoadCaseIDList(0)
iLoadCaseIDListToSet(0) = iLCIdentifierString
Dim iLCInSteps(0)
iLCInSteps(0) = 1
oFrameSelector.SetLoadCases iLCInSteps, iLoadCaseIDListToSet
```

3) Setting the frames to the frame selector

Note: One can use the SimResultsStep::GetFramesData() method to get the frames data for the step

```
Dim iFramesInLC(0)
iFramesInLC(0) = 2
Dim oGlobalLoadCaseIDNames()
Dim oLoadCaseIDListInStep()
```

```
Dim iFrameIndex(1)
iFrameIndex(0) = 0
iFrameIndex(1) = 1
oFrameSelector.SetFrames iFramesInLC, iFrameIndex
```

The following method must be called at the end after setting the Step, LC and Frame information.

```
oLocalSensor.FrameSelector = oFrameSelector
```

...

8. Sets the frame selection in the sensor

Following code sets the frame selection in the sensor.

```

...
oLocalSensor.FrameSelector = oFrameSelector
...
```

9. Sets the sensor parameters

Here we set the sensor parameters such as Max, Min, Absolute Max, Parameter for each step and Parameter for each frame or load case.

```

...
oLocalSensor.SetParameters True,True,True,False,False
```

The following steps shows how to set the Sum and Average parameters:

```
Dim oSensorOption As SimSensorOptions
Set oLocalSensor.GetItem("SimSensorOptions")

oSsensorOption.SetSumParameter True
oSsensorOption.SetAverageParameter True
...
```

10. Updates the sensor

Following code updates the sensor.

```
...
oLocalSensor.Update
...
```

11. Retrieves the sensor

This step shows how to retrieve the sensor from the results analysis case using the results set.

```
...
Dim oResultSet As SimResultsSet
Set oResultSet = oResultsAnalysisCase.GetSet("SensorSet")

Dim oRetrievedSensor As SimSensor

The sensor can be retrieved from the results set by following two ways:
a) Index:
Set oRetrievedSensor = oResultSet.Item(1)

b) Name:
Set oRetrievedSensor = oResultSet.Item("Sensor S.1")
...
```

12. Show annotation on the sensor

Following shows annotation on the sensor. The annotation contains information regarding the step, frame and parameter values

```
...
oRetrievedSensor.Annotation = True
...
```

Creating a sensor

This article explains how to open the results and create a sensor.

Before you begin: Note that:

- **Opening .3dxml file:** You should first launch CATIA and then import the CAAScdfeaResultsMultiLoadCase.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: [CAAScdfeaResultsSensorSource.htm](#)

This use case can be divided in 11 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Creates the sensor](#)
3. [Sets the attributes to the sensor](#)
4. [Sets the axis to the sensor](#)
5. [Creates the frame selection](#)
6. [Sets the frame selection in the sensor](#)
7. [Sets the sensor parameters](#)
8. [Updates the sensor](#)
9. [Show annotation on the sensor](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case.

[Retrieve the analysis case](#)

2. Creates the sensor

This step shows how to create a sensor from the results analysis case retrieved from the earlier step.

```
...
oLocalSensor = oResultsAnalysisCase.CreateSensor()
...
```

Creates the sensor by Id

Creates a sensor using field plot definition defined in XML description library files. It takes input as the ID of the Field Plot as specified in the XML file as sensor if this method is used no need to set the sensor attributes mentioned in Step 3.

we can use the `GetListOfFieldPlotIDs` of `ResultsAnalysisCase` to get the list of plot ids. This plot ids can be used to create the sensor.

```
...
oLocalSensor = oResultsAnalysisCase.CreateSensorByID (csSensorID)
...
```

3. Sets the attributes to the sensor

The following code sets the sensor attributes.

```
...
oLocalSensor.VariableType = ("S")
oLocalSensor.ProcessingType = ENUM.SimQuantity
oLocalSensor.Location = ENUM.SimCentroids
oLocalSensor.TransformType = ENUM.SimUserModelAxis
oLocalSensor.QuantityType = ("Tensor Component 11")
...
```

4. Sets the axis to the sensor

There are three ways in which we can set the axis as a transformation.

a) Using the axes available in the SIM: The exact name of the coordinate system needs to be specified.

```
...
oLocalSensor.TransformType = SimUserDefined
oLocalSensor.CoordinateSystem = "Orientation-Rib02--1-1"
...
```

b) The User Csys created in Physics results:

```
...
oResultsCsysSet = oResultsAnalysisCase.GetSet("UserCsysSet")
oUserCsys = oResultsCsysSet.Item(1)

oLocalSensor.TransformType = SimUserDefined
oAxisSystemAxis = oLocalSensor.GetItem("SimResultsAxisSystemAccess")

oAxisSystemAxis.Axis = oUserCsys
...
```

c) The model axes created at the product level: First the axis needs to be retrieved from the product, then set it to the sensor as below. Refer the sample script to see how to retrieve axis from the product.

```
...
oLocalSensor.TransformType = SimUserModelAxis

AxisSystems = MyPart.AxisSystems
AxisSystem = AxisSystems.Item(2)

oAxisSystemAxis = oLocalSensor.GetItem("SimResultsAxisSystemAccess")
oAxisSystemAxis.Axis = AxisSystem
...
```

Note: For setting the axis call the "TransformType" and "QuantityType" in the same order as specified in Step 3. The TransformType

5. Creates the frame selection

Here we create a frame selection object.

```
...
oFrameSelector = oResultsAnalysisCase.CreateFrameSelector()
...
```

6. Sets the frame selection attributes

In this step we set the frame selection attributes. More options for frames selection can be found at [Frames Selection](#)

a) Setting pre defined frame selection enumns in the sensor:

```
oFrameSelector.SymbolicFrameRange = SimAllFrames
```

b) Setting custom frame selection to the sensor:

1) Retrieving the Step ID and setting it to the frame selector
`oResultsSteps = ResultsAnalysisCase.ResultsSteps`

```
oResultsStep = oResultsSteps.Item(2)
```

```
sID, sStepName, sDescription = oResultsStep.GetIdentifier("", "", "")
lsStepID = [sID]
```

```
FrameSelector.StepList = lsStepID
```

2) Retrieving the Load case ID and setting it to the frame selector

Note: It is mandatory to set the Load Case, if the selected step has one.

The following method returns all the Load case ID avaialble in the case. The order of the list is the same as the user has created load case. One can get the indices of the Load case available in that step using GetLoadCases() method as shown below
`oResultsManager.GetLoadCaseIdentifier oGlobalLoadCaseIDList, oGlobalLoadCaseIDNames`

```
oResultsStep.GetLoadCases oLoadCaseIDListInStep

iLCIdentifierString = oGlobalLoadCaseIDList[0]
iLCInSteps = [1]
oFrameSelector.SetLoadCases (iLCInSteps, iLCIdentifierString)
```

3) Setting the frames to the frame selector

Note: One can use the `SimResultsStep::GetFramesData()` method to get the frames data for the step

```
iFramesInLC = [2]
iFrameIndex = [0,1]
oFrameSelector.SetFrames iFramesInLC, iFrameIndex
```

The following method must be called at the end after setting the Step, LC and Frame information.

```
oLocalSensor.FrameSelector = oFrameSelector
```

```
...
```

7. Sets the frame selection in the sensor

Following code sets the frame selection in the sensor.

```
...
```

```

oLocalSensor.FrameSelector = oFrameSelector
...

```

8. Sets the sensor parameters

Here we set the sensor parameters such as Max, Min, Absolute Max, Parameter for each step and Parameter for each frame or load case.

```

...
oLocalSensor.SetParameters (True,True,True,False,False)

The following steps shows how to set the Sum and Average parameters:

oSensorOption = GetItem("SimSensorOptions")

oSensorOption.SetSumParameter (True)
oSensorOption.SetAverageParameter (True)
...

```

9. Updates the sensor

Following code updates the sensor.

```

...
oLocalSensor.Update()
...

```

10. Show annotation on the sensor

Following shows annotation on the sensor. The annotation contains information regarding the step, frame and parameter values

```

...
oLocalSensor.Annotation = True
...

```

Creating a resultant sensor

This article explains how to open the results and create a resultant sensor.

Where to find the macro: [CAAScdfeaResultsResultantSensorSource.htm](#)

This use case can be divided in 6 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Creates the sensor](#)
3. [Sets step and frame](#)
4. [Sets the support for the sensor](#)
5. [Updates the sensor](#)
6. [Exports the sensor](#)
7. [Retrieves the sensor](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case.

[Retrieve the analysis case](#)

2. Creates the sensor

This step shows how to create a sensor from the results analysis case retrieved from the earlier step.

```

...
Dim oResultantSensor As SimResultantSensor
Set oResultantSensor = oResultsAnalysisCase.CreateResultantSensor
...

```

3. Sets step and frame

The following code sets the step and frame.

```

...
Dim oSteps As SimResultsSteps
Set oSteps = oResultsAnalysisCase.ResultsSteps

Dim oStep As SimResultsStep
Set oStep = oSteps.Item(1)

Dim osStepID As String
Dim ocsStepName As String
Dim ocsStepDescription As String

oStep.GetIdentifier osStepID, ocsStepName, ocsStepDescription
oResultantSensor.SetStepAndFrame osStepID, 1, 0
...

```

4. Sets the support for the sensor

In this step the support for the sensor is specified. The API and input for orphan and native simulations is different as explained below:

a) Orphan simulation: The SimSupport method should be used to set the support. It takes the input values as list of strings which are the names of the node sets.

```

...
Dim MySupports(0)
MySupports(0) = "Clamp1"

oResultantSensor.SimSupport = MySupports
...

```

b) Native simulation: The Support method is used to set the support. It takes input as list of features such as restraints, loads.

Find here [CAAScdfeaResultsNativeResultantSensorSource](#) the sample script.

```
...
Dim oDispatch As CATBaseDispatch
Set oDispatch = oPressure

Dim MySupports(0) As CATSafeArrayVariant
Set MySupports(0) = oDispatch

oResultantSensor.Support = MySupports
...
```

Important note: The above code where the CATSafeArrayVariant is used works fine in CATScript. But for VBScript and VBA, one must un-type the variable which calls the support method. Thus in the above code "oResultantSensor" variable must be un-typed. Thus in Step 2, the variable must be declared without its class type as stated below:

```
Dim oResultantSensor
```

For further details on usage of Arrays please refer to : [About Microsoft Automation Languages, Debug, and Compatibility](#)

5. Updates the sensor

Following code updates the sensor.

```
...
oResultantSensor.Update
...
```

6. Exports the sensor

The sensor can be exported to csv file either on disk or as a PLM document.

```
...
(a) Saving the file on disk: User can specify multiple sensors which will be exported in a single file.
```

```
Dim olsSensors(0) As CATSafeArrayVariant
Set olsSensors(0) = oResultantSensor

Dim icsFileName As String
icsFileName = "ExportedSensor"

Dim icsFileLocation As String
icsFileLocation = "C:\temp"

Dim ieFileType As SimFileType
ieFileType = SimXlsx

Note: The following boolean specified if the axis direction is to be exported

Dim bActivate As Boolean
bActivate = True

oResultsAnalysisCase.ExportSensorOutputToFile olsSensors, icsFileName, icsFileLocation, ieFileType, True, SimComma
```

(b) Saving as a PLM document: Only 1 sensor can be exported at a time.

```
Dim olsSensors(0) As CATSafeArrayVariant
Set olsSensors(0) = oResultantSensor

Dim icsFileName As String
icsFileName = "ExportedSensor"

Dim ieFileType As SimFileType
ieFileType = SimXlsx

Note: The following boolean specified if the axis direction is to be exported

Dim bActivate As Boolean
bActivate = True

oResultsAnalysisCase.ExportSensorOutputToPLM olsSensors, icsFileName, SimCsv, False, SimSemicolon
...
```

7. Retrieves the sensor

This step shows how to retrieve the sensor from the results analysis case using the results set.

```
...
Dim oResultSet As SimResultsSet
Set oResultSet = oResultsAnalysisCase.GetSet("ResultantSensorSet")

Dim oRetrievedSensor As SimResultantSensor

The sensor can be retrieved from the results set by following two ways:
a) Index:
Set oRetrievedSensor = oResultSet.Item(1)

b) Name:
Set oRetrievedSensor = oResultSet.Item("Resultant Sensor.1")
...
```

Creating a resultant sensor

This article explains how to open the results and create a resultant sensor.

Where to find the macro: [CAAScdfeaResultsResultantSensorSourcePython.htm](#)

This use case can be divided in 6 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Creates the sensor](#)
3. [Sets step and frame](#)
4. [Sets the support for the sensor](#)
5. [Updates the sensor](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case.

[Retrieve the analysis case](#)

2. Creates the sensor

This step shows how to create a sensor from the results analysis case retrieved from the earlier step.

```
...
oResultantSensor = oResultsAnalysisCase.CreateResultantSensor()
...
```

3. Sets step and frame

The following code sets the step and frame.

```
...
oSteps = oResultsAnalysisCase.ResultsSteps
oStep = oSteps.Item(1)

stepId, stepName, stepDescription = oStep.GetIdentifier (), (), ()
oResultantSensor.SetStepAndFrame (stepId, 1, 0)
...
```

4. Sets the support for the sensor

In this step the support for the sensor is specified. The API and input for orphan and native simulations is different as explained below:

a) Orphan simulation: The SimSupport method should be used to set the support. It takes the input values as list of strings which are the names of the node sets.

```
...
MySupports = ["Clamp1"]
oResultantSensor.SimSupport = (MySupports)
...
```

b) Native simulation: The Support method is used to set the support. It takes input as list of features such as restraints, loads.

Find here [CAAScdfeaResultsNativeResultantSensorSourcePython](#) the sample script.

```
...
oDispatch = oPressure
MySupports(0) = oDispatch
oResultantSensor.Support = (MySupports)
...
```

Important note: The above code where the CATSafeArrayVariant is used works fine in CATScript. But for VBScript and VBA, one must un-type the variable which calls the support method. Thus in the above code "oResultantSensor" variable must be un-typed. Thus in Step 2, the variable must be declared without its class type as stated below:

For further details on usage of Arrays please refer to : [About Microsoft Automation Languages, Debug, and Compatibility](#)

5. Updates the sensor

Following code updates the sensor.

```
...
oResultantSensor.Update
...
```

Creating a frequency sensor

This article explains how to open the results and create a frequency sensor.

Where to find the macro: [CAAScdfeaResultsFrequencySensorSource.htm](#)

This use case can be divided in 7 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Creates a frequency sensor](#)
3. [Creates the frame selection](#)
4. [Sets the frame selection attributes](#)
5. [Sets the frame selection in the sensor](#)
6. [Updates the sensor](#)
7. [Retrieves the sensor](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case.

[Retrieve the analysis case](#)**2. Creates a frequency sensor**

This step shows how to create a sensor from the results analysis case retrieved from the earlier step.

```
...
Dim oFrequencySensor As SimFrequencySensor
Set oFrequencySensor = oResultsAnalysisCase.CreateFrequencySensor
...
```

3. Creates the frame selection

Here we create a frame selection object.

```
...
Dim oFrameSelector As SimFramesSelection
Set oFrameSelector = oResultsAnalysisCase.CreateFrameSelector
...
```

4. Sets the frame selection attributes

In this step we set the frame selection attributes. More options for frames selection can be found at [Frames Selection](#)

```
...
oFrameSelector.SymbolicFrameRange = SimFrequencyFrames
...
```

5. Sets the frame selection in the sensor

Following code sets the frame selection in the sensor.

```
...
oFrequencySensor.FrameSelector = oFrameSelector
...
```

6. Updates the sensor

Following code updates the sensor.

```
...
oFrequencySensor.Update
...
```

7. Retrieves the sensor

This step shows how to retrieve the sensor from the results analysis case using the results set.

```
...
Dim oResultSet As SimResultsSet
Set oResultSet = oResultsAnalysisCase.GetSet("FrequencySensorSet")

Dim oRetrievedSensor As SimFrequencySensor

The sensor can be retrieved from the results set by following two ways:
a) Index:
Set oRetrievedSensor = oResultSet.Item(1)

b) Name:
Set oRetrievedSensor = oResultSet.Item("Frequency Sensor.1")
...
```

Creating a frequency sensor

This article explains how to open the results and create a frequency sensor.

Where to find the macro: [CAASedfeaResultsFrequencySensorSourcePython.htm](#)

This use case can be divided in 7 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Creates a frequency sensor](#)
3. [Creates the frame selection](#)
4. [Sets the frame selection attributes](#)
5. [Sets the frame selection in the sensor](#)
6. [Updates the sensor](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case.

[Retrieve the analysis case](#)**2. Creates a frequency sensor**

This step shows how to create a sensor from the results analysis case retrieved from the earlier step.

```
...
oFrequencySensor = oResultsAnalysisCase.CreateFrequencySensor
...
```

3. Creates the frame selection

Here we create a frame selection object.

```
...
oFrameSelector = oResultsAnalysisCase.CreateFrameSelector()
...
```

4. Sets the frame selection attributes

In this step we set the frame selection attributes. More options for frames selection can be found at [Frames Selection](#)

```
...
oFrameSelector.SymbolicFrameRange = ENUM.SimFrequencyFrames
...
```

5. Sets the frame selection in the sensor

Following code sets the frame selection in the sensor.

```
...
oFrequencySensor.FrameSelector = oFrameSelector
...
```

6. Updates the sensor

Following code updates the sensor.

```
...
oFrequencySensor.Update()
...
```

Creating a buckle mode sensor

This article explains how to open the results and create a buckle mode sensor.

Where to find the macro: [CAAScdfeaResultsBuckleModeSensorSource.htm](#)

This use case can be divided in 7 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Creates a buckle mode sensor](#)
3. [Creates the frame selection](#)
4. [Sets the frame selection attributes](#)
5. [Sets the frame selection in the sensor](#)
6. [Updates the sensor](#)
7. [Retrieves the sensor](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case. You can use `VB_Buckle_Mode_Sensor.3dxml` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

[Retrieve the analysis case](#)

2. Creates a buckle mode sensor

This step shows how to create a sensor from the results analysis case retrieved from the earlier step.

```
...
Dim oBuckleModeSensor As SimBuckleModeSensor
Set oBuckleModeSensor = oResultsAnalysisCase.CreateBuckleModeSensor
...
```

3. Creates the frame selection

Here we create a frame selection object.

```
...
Dim oFrameSelector As SimFramesSelection
Set oFrameSelector = oResultsAnalysisCase.CreateFrameSelector
...
```

4. Sets the frame selection attributes

In this step we set the frame selection attributes. More options for frames selection can be found at [Frames Selection](#)

```
...
oFrameSelector.SymbolicFrameRange = SimAllFrames
...
```

5. Sets the frame selection in the sensor

Following code sets the frame selection in the sensor.

```
...
oBuckleModeSensor.FrameSelector = oFrameSelector
...
```

6. Updates the sensor

Following code updates the sensor.

```
...
oBuckleModeSensor.Update
...
```

7. Retrieves the sensor

This step shows how to retrieve the sensor from the results analysis case using the results set.

```
...
Dim oResultSet As SimResultsSet
Set oResultSet = oResultsAnalysisCase.GetSet("BuckleModeSensorSet")

Dim oRetrievedSensor As SimBuckleModeSensor

The sensor can be retrieved from the results set by following two ways:
a) Index:
Set oRetrievedSensor = oResultSet.Item(1)

b) Name:
Set oRetrievedSensor = oResultSet.Item("Buckle Mode Sensor.1")
...
```

Creating a buckle mode sensor

This article explains how to open the results and create a buckle mode sensor.

Where to find the macro: [CAAScdfeaResultsBuckleModeSensorSourcePython.htm](#)

This use case can be divided in 6 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Creates a buckle mode sensor](#)
3. [Creates the frame selection](#)
4. [Sets the frame selection attributes](#)
5. [Sets the frame selection in the sensor](#)
6. [Updates the sensor](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case. You can use `VB_Buckle_Mode_Sensor.3dxml` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

[Retrieve the analysis case](#)

2. Creates a buckle mode sensor

This step shows how to create a sensor from the results analysis case retrieved from the earlier step.

```
...
oBuckleModeSensor = oResultsAnalysisCase.CreateBuckleModeSensor()
...
```

3. Creates the frame selection

Here we create a frame selection object.

```
...
oFrameSelector = oResultsAnalysisCase.CreateFrameSelector()
...
```

4. Sets the frame selection attributes

In this step we set the frame selection attributes. More options for frames selection can be found at [Frames Selection](#)

```
...
oFrameSelector.SymbolicFrameRange = ENUM.SimAllFrames
...
```

5. Sets the frame selection in the sensor

Following code sets the frame selection in the sensor.

```
...
oBuckleModeSensor.FrameSelector = oFrameSelector
...
```

6. Updates the sensor

Following code updates the sensor.

```
...
oBuckleModeSensor.Update()
...
```

Creating a history sensor

This article explains how to open the results and create a history sensor.

Where to find the macro: [CAAScdfeaResultsHistorySensorSource.htm](#)

This use case can be divided in 9 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Creates the history sensor](#)
3. [Get the step Id ,name and step description](#)
4. [Sets the attributes to the history sensor](#)
5. [Sets the support for the history sensor](#)
6. [Sets step, frame and load case information](#)

7. [Sets the sensor parameters](#)
8. [Updates the history sensor](#)
9. [Retrieves the sensor parameter values](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case. You can use `VB_HistoryPlot.SMASim` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdFeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-R installed.

Import the SMASim file. Click on the link below for details about how to retrieve the analysis case.

[Retrieve the analysis case](#)

2. Creates the history sensor

This step shows how to create a history sensor factory and history sensor from the results analysis case retrieved from the earlier step.

```
...  
Dim oHistorySensorFactory As SimHistorySensorFactory  
Set oHistorySensorFactory = oResultsAnalysisCase.GetItem("SimHistorySensorFactory")  
  
Dim oHistorySensor As SimHistorySensor  
Set oHistorySensor = oHistorySensorFactory.CreateSensor  
...
```

3. Get the step Id ,name and step description

This step shows how to get the step Id, name and step description.

```
...  
dim oSteps As SimResultsSteps  
Set oSteps = oResultsAnalysisCase.ResultsSteps  
  
dim oResultsStep As SimResultsStep  
Set oResultsStep = oSteps.Item(1)  
  
dim stepId As String  
dim stepName As String  
dim stepDescription As String  
  
oResultsStep.GetIdentifier stepId, stepName, stepDescription  
...
```

4. Sets the attributes to the history sensor

The following code sets the history sensor attributes. In this step we set the history sensor attributes. More options for history sensor attributes can be found at [HistoryplotEnums](#).

```
...  
oHistorySensor.Variable = "UT"  
oHistorySensor.ComponentQuantity = SimVector_Component_1  
  
Dim ieComplexValue As SimComplexValues  
ieComplexValue = SimEnvelopeMax  
  
Dim iValueAtAngle As double  
iValueAtAngle = 30  
  
oHistorySensor.SetComplex ieComplexValue, iValueAtAngle  
...
```

5. Sets the support for the history sensor.

The following code sets support for creating the history sensor such as regions and sub-regions. If user wants to select the whole model, no need to call this method

```
...  
Dim MySupports(0)  
MySupports(0) = "INP2SIM_NSET_Nodal_Probe1"  
  
Dim MySupports1(0)  
MySupports1(0) = 3  
  
Dim MySupports2(2)  
MySupports2(0) = "Node 4"  
MySupports2(1) = "Node 87"  
  
oHistorySensor.SetSupport MySupports, MySupports1, MySupports2  
oHistorySensor.SetStepAndFrame stepId, 0, 0  
...
```

6. Sets the step, frame and load case information

Following code sets the step, frame and load case information.

`stepId` is the persistent ID of the step.

The Frame index starts from 1. This option is needed only in case of multiple load cases. In the following code we have used 0 as the frame index.

The load case index starts from 1. It is needed if a load case is present in the step. In the following code we have used 0 as the load case index.

```
...  
oHistorySensor.SetStepAndFrame stepId, 0, 0  
...
```

Using following code we can set all available time based steps.

```
...  
oHistorySensor.SetAllTimeBasedSteps
```

...

7. Sets the history sensor parameters

Following code sets the history sensor parameters such as max, min, absolute max.

```
...
    oHistorySensor.SetParameters True, True, True
...
```

```
The following line creates parameter for the last value. The last parameter be created only if a single sub-region is selected as
...
    oHistorySensor.LastValue = True
...
```

8. Updates the history sensor

Following code updates the history sensor. This method needs to be called at the end after setting the attributes.

```
...
    oHistorySensor.Update
...
```

9. Retrieves the sensor parameter values.

This step retrieves the minimum and maximum values.

```
...
    Dim oKnowServices As KnowledgeServices
    Set oKnowServices = CATIA.GetSessionService("KnowledgeServices")

    Dim oKnowCollection
    Set oKnowCollection = oKnowServices.GetKnowledgeCollection(oHistorySensor, kweParametersType )

    Dim oParamMinValue As String
    oParamMinValue = oKnowCollection.Item(1).Value

    Dim oParamMaxValue As String
    oParamMaxValue = oKnowCollection.Item(2).Value
...

```

Creating a history sensor

This article explains how to open the results and create a history sensor.

Where to find the macro: [CAAScdFeaResultsHistorySensorSourcePython.htm](#)

This use case can be divided in 9 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Creates the history sensor](#)
3. [Get the step Id ,name and step description](#)
4. [Sets the attributes to the history sensor](#)
5. [Sets the support for the history sensor](#)
6. [Sets step, frame and load case information](#)
7. [Sets the sensor parameters](#)
8. [Updates the history sensor](#)
9. [Retrieves the sensor parameter values](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case. You can use `VB_HistoryPlot.SMASim` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdFeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-R installed.

Import the SMASim file. Click on the link below for details about how to retrieve the analysis case.

[Retrieve the analysis case](#)

2. Creates the history sensor

This step shows how to create a history sensor factory and history sensor from the results analysis case retrieved from the earlier step.

```
...
    oHistorySensorFactory = oResultsAnalysisCase.GetItem("SimHistorySensorFactory")
    oHistorySensor = oHistorySensorFactory.CreateSensor
...

```

3. Get the step Id ,name and step description

This step shows how to get the step Id,name and step description.

```
...
    oSteps = oResultsAnalysisCase.ResultsSteps
    oStep = oSteps.Item(1)
    stepId, stepName, stepDescription = oStep.GetIdentifier (), (), ()
...

```

4. Sets the attributes to the history sensor

The following code sets the history sensor attributes.In this step we set the history sensor attributes.More options for history sensor attributes can be found at [HistoryplotEnums](#)

```
...
    oHistorySensor.Variable = "UT"
    oHistorySensor.ComponentQuantity = ENUM.SimVector_Component_1
    ieComplexValue = ENUM.SimEnvelopeMax
    iValueAtAngle = 30
    oHistorySensor.SetComplex (ieComplexValue, iValueAtAngle)
...

```

5. Sets the support for the history sensor.

The following code sets support for creating the history sensor such as regions and sub-regions. If user wants to select the whole model, no need to call this method

```
...
    MySupports = ["INP2SIM_NSET_Nodal_Probe1"]
    MySupports1 = [3]
    MySupports2 = ["Node 4", "Node 87"]
    oHistorySensor.SetSupport (MySupports, MySupports1, MySupports2)
...

```

6. Sets the step, frame and load case information

Following code sets the step,frame and load case information.

stepId is the persistent ID of the step.

The Frame index starts from 1. This option is needed only in case of multiple load cases.In the following code we have used 0 as the frame index.

The load case index starts from 1. It is needed if a load case is present in the step.In the following code we have used 0 as the load case index.

```
...
    oHistorySensor.SetStepAndFrame (stepId, 0, 0)
...

```

Using following code we can set all available time based steps.

```
...
    oHistorySensor.SetAllTimeBasedSteps()
...

```

7. Sets the history sensor parameters

Following code sets the history sensor parameters such as max, min, absolute max.

```
...
    oHistorySensor.SetParameters (True, True, True)
...

```

The following line creates parameter for the last value. The last parameter be created only if a single sub-region is selected as
`... oHistorySensor.LastValue = True`
`...`

8. Updates the history sensor

Following code updates the history sensor. This method needs to be called at the end after setting the attributes.

```
...
    oHistorySensor.Update()
...

```

9. Retrieves the sensor parameter values.

This step retrieves all the parameters. Here it is maximum and minimum values.

```
...
    oKnowServices = CATIA.GetSessionService("KnowledgeServices")
    oKnowCollection = oKnowServices.GetKnowledgeCollection(oHistorySensor, ENUM.kweParametersType )
    oParamMinValue = oKnowCollection.Item(1).Value
    oParamMaxValue = oKnowCollection.Item(2).Value
...

```

Creating a strain gauge sensor

This article explains how to open the results and create a strain gauge sensor.

Where to find the macro: [CAAScdfeaResultsStrainGaugeSensorSource.htm](#)

This use case can be divided in 9 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Creates the strain gauge sensor](#)
3. [Get the step Id ,name and step description](#)
4. [Sets step, frame and load case information](#)
5. [Sets the attributes to the sensor](#)
6. [Sets the gauge attributes](#)
7. [Updates the sensor](#)
8. [Retrieves the sensor parameter values](#)
9. [Exports the sensor gauge definition](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case. You can use CAAScdfeaSimResults.SMASim file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Import the SMASim file. Click on the link below for details about how to retrieve the analysis case.

[Retrieve the analysis case](#)

2. Creates the strain gauge sensor

This step shows how to create a strain gauge sensor factory and strain gauge sensor from the results analysis case retrieved from the earlier step.

```
...
Dim oStrainGaugeSensorFactory As SimStrainGaugeSensorFactory
Set oStrainGaugeSensorFactory = oResultsAnalysisCase.GetItem("SimStrainGaugeSensorFactory")

Dim oStrainGaugeSensor As SimStrainGaugeSensor
Set oStrainGaugeSensor = oStrainGaugeSensorFactory.CreateSensor

...
```

3. Get the step Id ,name and step description

This step shows how to get the step Id, name and step description.

```
...
dim oSteps As SimResultsSteps
Set oSteps = oResultsAnalysisCase.ResultsSteps

dim oResultsStep As SimResultsStep
Set oResultsStep = oSteps.Item(1)

dim stepId As String
dim stepName As String
dim stepDescription As String

oResultsStep.GetIdentifier stepId, stepName, stepDescription

...
```

4. Sets the step, frame and load case information

Following code sets the step, frame and load case information.

stepId is the persistent ID of the step.

The Frame index starts from 1. This option is needed only in case of multiple load cases. In the following code we have used 0 as the frame index.

The load case index starts from 1. It is needed if a load case is present in the step. In the following code we have used 0 as the load case index.

```
...
oStrainGaugeSensor.SetStepAndFrame stepId, 2, 0

...
```

5. Sets the attributes to the sensor.

The following code sets the variable type. The available type are SimStress, SimStrainFromStrain and SimStrainFromDisplacement

```
...
oStrainGaugeSensor.VariableType = SimStress

...
```

6. Sets the gauge attributes

Following code sets the size, origin and orientation of gauges. We are creating 2 gauges here.

```
...
Note:The length, width and tolerance must be specified in MKS.
oStrainGaugeSensor.SetSize True, 0.01, 0.008
oStrainGaugeSensor.GaugeTolerance = 0.005

Note:The origin location must be specified in mm. The axis direction is unitless.
For SetOriginLocations, the first parameter is an unique index. The other parameters are the X,Y and Z values of origin.
For SetAxisDirections, the first 3 parameters are the X directions(gauge) and the other 3 parameters are the Z directions(set).

oStrainGaugeSensor.SetOriginLocations 1, 20, 70, 50
oStrainGaugeSensor.SetAxisDirections 0.12, 0.99, 0, 0, -0, 1
oStrainGaugeSensor.SetOriginLocations 2, 20, -80, 50
oStrainGaugeSensor.SetAxisDirections 0.273, -0.96, 0, 0, -0, 1
oStrainGaugeSensor.ShowGlyph = True

...
```

7. Updates the sensor

Following code updates the sensor. This method needs to be called at the end after setting the attributes.

```

oStrainGaugeSensor.Update
...

```

8. Retrieves the sensor gauge values.

This step retrieves the gauge values.

```

...
Dim oKnowServices As KnowledgeServices
Set oKnowServices = CATIA.GetSessionService("KnowledgeServices")

Dim oKnowCollection
Set oKnowCollection = oKnowServices.GetKnowledgeCollection(oStrainGaugeSensor, kweParametersType)

Dim oValue1 As String
oValue1 = oKnowCollection.Item(1).Value

Dim oValue2 As String
oValue2 = oKnowCollection.Item(2).Value
...

```

9. Exports the sensor gauge definition.

This step Exports the sensor gauge definition to csv file.

```

...
oStrainGaugeSensor.Export "VSG_Export", "C:\temp", SimCsv
...

```

Exporting field plots

This article explains how to export the plot to a file or a PLM document. The file is exported in .csv format

Where to find the macro: [CAAScdfeaResultsExportPlotSource.htm](#)

This use case can be divided in following steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Retrieves the field plot from existing list](#)
3. [Export the plot to file](#)
4. [Exports the plot to a PLMDocument](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case. You can use CAAScdfeaResults.3dxml file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

- [Retrieve the analysis case](#)
- 2. Retrieves the field plot**

This step shows how to retrieve the field plot from existing list of plots.

```

...

```

```

Dim oFieldPlots As SimResultsSet
Set oFieldPlots = oResultsAnalysisCase.GetSet("FieldPlots")

Dim oFieldPlot As SimFieldPlot
Set oFieldPlot = oFieldPlots.Item(2)
...

```

3. Export the plot to file

This step shows how to export the plot to file.

```

...

```

```

Dim oExportPlot As SimExportField
Set oExportPlot = oResultsAnalysisCase.CreateExportFieldFromPlot(oFieldPlot)

Dim iFileName As String
iFileName = "Field_Plot_Export"

Dim iFileLocation As String
iFileLocation = "C:\temp"

Dim iDeformed As Integer
iDeformed = 1

Dim iExportOnExteriorValuesOnly As Integer
iExportOnExteriorValuesOnly = 0

' The ExcludeNullValues() method can be called if one wants to exclude the NULL values.
' It must be called before the WriteToFile() or CreateFeatureLinkedDocument() methods.

oExportPlot.ExcludeNullValues = True

oExportPlot.WriteToFile iFileName, iFileLocation, iDeformed, iExportOnExteriorValuesOnly
...

```

4. Exports the plot to a PLM Document

The following code creates new Tabular Export Field feature and PLM document which will be linked to this feature

```
...
iFileName = "My_Export_Document"
oExportPlot.CreateFeatureLinkedDocument iFileName, iDeformed, iExportOnExteriorValuesOnly
...
```

Setting animation options and creating video

This article explains how to set animation options and create a movie file.

Where to find the macro: [CAAScdfeaResultsAnimationOptionsSource.htm](#)

This use case can be divided in 8 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Retrieves the animation options object](#)
3. [Retrieves the available animation options](#)
4. [Sets the animation options](#)
5. [Retrieves the animation video object](#)
6. [Retrieves the available codec options](#)
7. [Sets the video options](#)
8. [Creates the video](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case. You can use `VB_HistoryPlot.SMASim` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdfeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

On opening the model, Select yes option from the dialogue to activate the results.

[Retrieve the analysis case](#)

2. Retrieves the animation options object

This step retrieves the animation options object from the results analysis case. Each case can have its own animation option object

```
...
Dim oAnimationOptions As SimResultsAnimationOptions
oAnimationOptions = oResultsAnalysisCase.GetItem("SimResultsAnimationOptions")
...
```

3. Retrieves the available animation types

This step shows how to get the available animation types for the current plot. It will be a list of strings.

```
...
Dim ocsAnimTypes()
oAnimationOptions.GetAvailableAnimationTypes ocsAnimTypes
...
```

4. Sets the animation types

It is mandatory to call the `SetType` method at the beginning and `Update` method at the end after all the values are set.

```
...
a) Its is mandatory to set the type. User can set attributes for all the animation types
```

Following are the different animation types. The availability of the type depends on the current plot type. The `GetAvailableAnimationTypes()` method can be called to get the list of available animation types.

- 1) SimScaleFactor
- 2) SimTimeHistory
- 3) SimLoopOverModes
- 4) SimScanSingleFrame
- 5) SimTimeHistoryAndLoopover
- 6) SimFrameBased
- 7) SimStreamline
- 8) SimEventSeries
- 9) SimHarmonic

```
Dim oAnimType As SimAnimationTypes
oAnimType = SimTimeHistory
oAnimationOptions.SetType oAnimType
```

b) Frame selector needs to be set only for time history and loop over mode

```
Dim FrameSelector As SimFramesSelection
Set FrameSelector = oResultsAnalysisCase.CreateFrameSelector
FrameSelector.SymbolicFrameRange = SimAllFrames
oAnimationOptions.SetFrameSelector FrameSelector
```

c) Specifies whether to animate over all the frames or with specific intervals

```
Dim oSamplingType As SimSamplingRateTypes
oSamplingType = SimAllSelectedFrames
Dim oSamplingFilterType As SimSamplingFilterTypes
oSamplingFilterType = SimTotalFrames
oAnimationOptions.SetSamplingRate oSamplingType, oSamplingFilterType, 10
```

d) Specifies the speed of animation

```

Dim eTargetSpeedType As SimTargetSpeedTypes
eTargetSpeedType = SimTargetSpeedMax
oAnimationOptions.SetTargetSpeed eTargetSpeedType, 10

e) Updates the options. It has to be called at the end
oAnimationOptions.Update

...

```

5. Retrieves the animation video object.

The following code retrieves the animation video object from the SimResultsManager.

```

...
Dim oAnimationVideo As SimResultsAnimationVideo
Set oAnimationVideo = oRootRepRes.GetItem("SimResultsAnimationVideo")
...

```

6. Retrieves the available codec options

The codec options may be system specific. The different codecs are IntelliYUV, MicrosoftVideo1, FullFrames(compressed), "H.264 / MPEG-4 AVC"

```

...
Dim lsCodecs()
oAnimationVideo.GetListOfCodecOptions lsCodecs
...

```

1.

Sets the video options

Following code sets the video options.

```

...
a) The following enums are avaialble for SimAnimPlaybackTypes in SMAIMpaAnimationOptionsEnums:
   SimPlaybackFramesPerSec,
   SimPlaybackTotalTime

Dim oPlaybackType As SimAnimPlaybackTypes
oPlaybackType = SimPlaybackTotalTime
oAnimationVideo.SetPlayback oPlaybackType, 10

b) The codecs can be set as below.

  i) Microsoft Video 1 codec:
     Dim csCodec As String
     csCodec = "Microsoft Video 1"
     oAnimationVideo.SetCodec csCodec
     oAnimationVideo.SetCompressionOptions 88, True, 1, 100

  ii) H.264 / MPEG-4 AVC codec:
     Dim csCodec As String
     csCodec = "H.264 / MPEG-4 AVC"
     oAnimationVideo.SetCodec csCodec
     oAnimationVideo.SetRayTraceOptions 5, True, 1, True, True, True, 20

  iii) IntelliYUV and FullFrames(compressed) codecs:
     Dim csCodec As String
     csCodec = "FullFrames(compressed)"

c) Setting the XY plot options. Only the activated or opened history plot will be added in the video.
oAnimationVideo.SetXYPlotOptions True, 40, 40, 0, 100
...
```

7. Creates the video

Following code sets the path and file name to create the video.

```

...
Dim ofilePath As String
Dim ofileName As String
ofilePath = "D:\temp"
ofileName = "HistoryAnimationMovie.avi"
oResAnimVideo.CreateMovie ofilePath, ofileName
...
```

Setting animation options and creating video

This article explains how to set animation options and create a movie file.

Where to find the macro: [CAAScdfeaResultsAnimationOptionsSourcePython.htm](#)

This use case can be divided in 8 steps:

1. [Opens the document and retrieves the results analysis case](#)
2. [Retrieves the animation options object](#)
3. [Retrieves the available animation options](#)
4. [Sets the animation options](#)
5. [Retrieves the animation video object](#)
6. [Retrieves the available codec options](#)
7. [Sets the video options](#)
8. [Creates the video](#)

1. Opens the document and retrieves the results analysis case

As a first step, the following code will show how to open the simulation document and retrieve the results analysis case. You can use `VB_HistoryPlot.SMASim` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdFeaResults\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

On opening the model, Select yes option from the dialogue to activate the results.

[Retrieve the analysis case](#)

2. Retrieves the animation options object

This step retrieves the animation options object from the results analysis case. Each case can have its own animation option object

```
...
oAnimationOptions = oResultsAnalysisCase.GetItem("SimResultsAnimationOptions")
...
```

3. Retrieves the available animation types

This step shows how to get the available animation types for the current plot. It will be a list of strings.

```
...
ocsAnimTypes = oAnimationOptions.GetAvailableAnimationTypes(())
...

```

4. Sets the animation types

It is mandatory to call the `SetType` method at the beginning and `Update` method at the end after all the values are set.

```
...
a) Its is mandatory to set the type. User can set attributes for all the animation types
```

Following are the different animation types. The availability of the type depends on the current plot type.
The `GetAvailableAnimationTypes()` method can be called to get the list of available animation types.

- 1) SimScaleFactor
- 2) SimTimeHistory
- 3) SimLoopOverModes
- 4) SimScanSingleFrame
- 5) SimTimeHistoryAndLoopover
- 6) SimFrameBased
- 7) SimStreamline
- 8) SimEventSeries
- 9) SimHarmonic

```
oAnimType = ENUM.SimTimeHistory
oAnimationOptions.SetType (oAnimType)
```

b) Frame selector needs to be set only for time history and loop over mode

```
FrameSelector = oResultsAnalysisCase.CreateFrameSelector()
FrameSelector.SymbolicFrameRange = ENUM.SimAllFrames
oAnimationOptions.SetFrameSelector (FrameSelector)
```

c) Specifies whether to animate over all the frames or with specific intervals

```
oSamplingType = ENUM.SimAllSelectedFrames
oSamplingFilterType = ENUM.SimTotalFrames
oAnimationOptions.SetSamplingRate (oSamplingType, oSamplingFilterType, 10)
```

d) Specifies the speed of animation

```
eTargetSpeedType = ENUM.SimTargetSpeedMax
oAnimationOptions.SetTargetSpeed (eTargetSpeedType, 10)
```

e) Updates the options. It has to be called at the end

```
oAnimationOptions.Update()
```

```
...
```

5. Retrieves the animation video object.

The following code retrieves the animation video object from the `SimResultsManager`.

```
...
oAnimationVideo = oRootRepRes.GetItem("SimResultsAnimationVideo")
...
```

6. Retrieves the available codec options

The codec options may be system specific. The different codecs are IntelIYUV, MicrosoftVideo1, FullFrames(compressed), "H.264 / MPEG-4 AVC"

```
...
```

```

lsCodecs = []
lsCodecs = oAnimationVideo.GetListOfCodecOptions (lsCodecs)
...

```

7. Sets the video options

Following code sets the video options.

```

...
a) The following enums are available for SimAnimPlaybackTypes in SMAIMpaAnimationOptionsEnums:
    SimPlaybackFramesPerSec,
    SimPlaybackTotalTime

oPlaybackType = ENUM.SimPlaybackTotalTime
oAnimationVideo.SetPlayback (oPlaybackType, 10)

b) The codecs can be set as below.

i) Microsoft Video 1 codec:
    csCodec = "Microsoft Video 1"
    oAnimationVideo.SetCodec (csCodec)
    oAnimationVideo.SetCompressionOptions (88, True, 1, 100)

ii) H.264 / MPEG-4 AVC codec:
    csCodec = "H.264 / MPEG-4 AVC"
    oAnimationVideo.SetCodec (csCodec)
    oAnimationVideo.SetRayTraceOptions (5, True, 1, True, True, True, 20)

iii) IntelIYUV and FullFrames(compressed) codecs:
    csCodec = "FullFrames(compressed)"

c) Setting the XY plot options. Only the activated or opened history plot will be added in the video.
oAnimationVideo.SetXYPlotOptions (True, 40, 40, 0, 100)

...

```

8. Creates the video

Following code sets the path and file name to create the video.

```

...
ofilePath = "D:\\temp"
fileName = "HistoryAnimationMovie.avi"
oResAnimVideo.CreateMovie (ofilePath, fileName)

...

```

Simulation Interchange Model Overview

Technical Article

Abstract

This technical article presents an overview of the Simulation Interchange Model.

- [What is the Simulation Interchange Model \(SIM\)?](#)
- [SIM Results Python API](#)
- [SIM API Features](#)
- [SIM Data Model](#)
- [Limitations](#)
- [Accessing the SIM document generated within the 3DEXPERIENCE platform](#)

What is the Simulation Interchange Model (SIM)?

The Simulation Interchange Model (SIM) is the SIMULIA data storage or exchange mechanism used by 3DEXPERIENCE SIMULIA applications. It is a database designed to handle large volumes of data efficiently. It supports both model and results data.

The database access starts with accessing a file named .SMAManifest. This is an xml file that groups other files that make up the database; binary files with extensions .SMAFocus and .SMABulk.

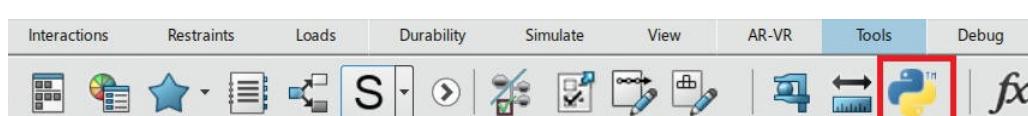
A SIM document can be generated by either a 3DEXPERIENCE simulation run or by a standalone Abaqus execution with resultsFormat SIM option. This SIM document can be read, accessed or modified using the SIM results API.

SIM Results Python API

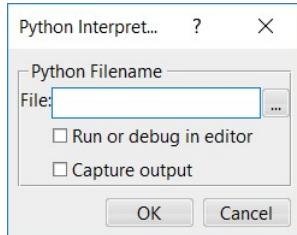
A SIM document is capable of storing both model and results data. The results SIM document can be visualized in the Physics Results Explorer app. While results documents are usually generated by SIMULIA apps, they can also be generated, modified, read using SIM results API, available in both C++ and Python. This section discusses the SIM Python API.

The SIM Results API has two modules- simResultsReader and simResultsWriter that help reading and writing results.

A script generated using these modules can be run within the 3DEXPERIENCE platform using the Python button in the Tools section of any SIMULIA app as shown below

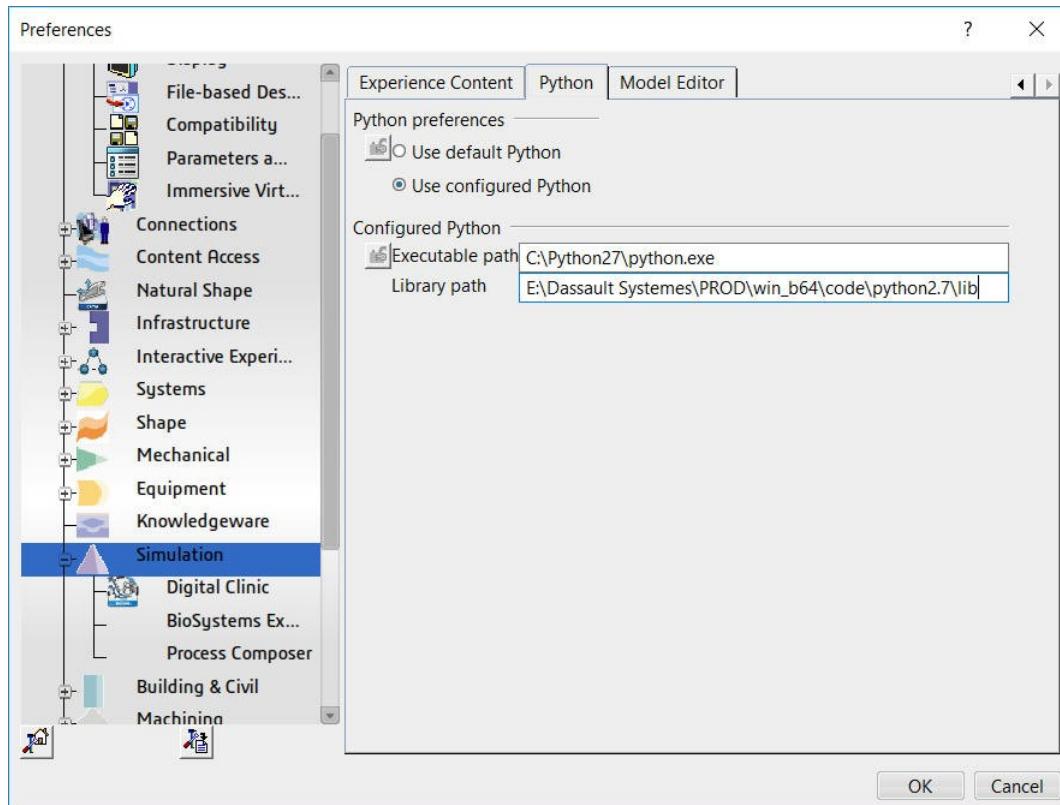


After clicking the Python button, you choose the Python file as shown below



By default, the Python interpreter is the 3DEXPERIENCE Python interpreter. You can also use Python downloaded from internet by following the steps below. Only Python 2 is supported.

1. Download and install the internet Python
2. In 3DX, go to Preferences -> Simulation -> Python
 - o Change the **Python preferences** to **Use configured Python**
 - o Specify the **Executable path** and **Library path** for the configured Python as shown below.
 - o The **Executable path** should be the path of the internet Python executable's path and the **Library path** should be the path which contains the Abaqus shared libraries.



3. Each script should have the following line at the beginning of the script:

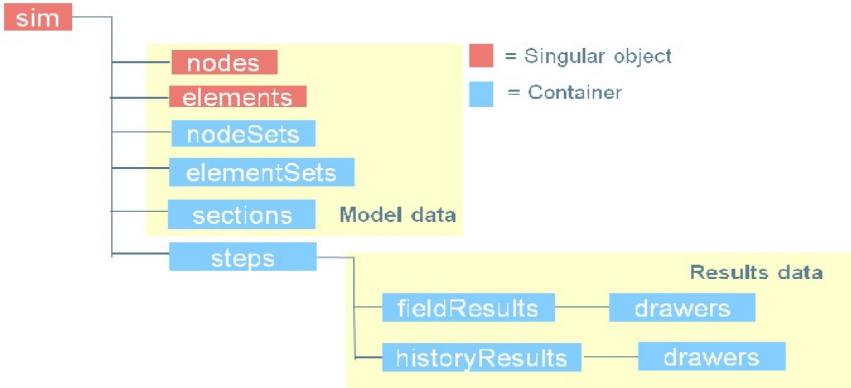
```
...
import sys
sys.path += ['E:\\Dassault Systemes\\PROD\\win_b64\\code\\python2.7\\lib']
#where 'E:\\Dassault Systemes\\PROD\\win_b64\\code\\python2.7\\lib' is the Python library path
...
```

SIM API Features

Some of the main features in the SIM API are:

- Uses Numpy arrays for efficiency.
- Throws clear exceptions during writing and reading phase.
- Supports Unicode characters.
- Uses very few classes as most of the methods are directly under reader and writer classes.
- Supports multiple field classes per output location.
- Provides functionality to read or write basic model data (nodes, elements, materials, sections etc.) and supports full results data.

SIM Data Model



Limitations

Some of the limitations in the SIM API are:

- The API does not provide the ability to read and write many model attributes such as loads, BCs, interactions, material properties, surfaces, etc. The SIM Python reference API showcases the coverage attributes.
- There are no methods to add part and assembly information using the SIM results writer API.

Accessing the SIM document generated within the 3DEXPERIENCE platform

The following snippet helps you access an SMAManifest file from a scenario object.

```
...
import com3dx
import simResultsReader

#Inside scenario app
CATIA = com3dx.get3dxClient()
myEditor = CATIA.ActiveEditor
myProdService = myEditor.GetService("PLMProductService")
mySimuObj = myProdService.EditedContent.Item(1)
myProduct = mySimuObj.Model

#Get FEM root
mySimuService = CATIA.GetSessionService("SimSimulationService")
myRootOccurrence = mySimuService.GetProductRootOccurrence(mySimuObj)
myFemRoot = None
myRef = myRootOccurrence.ReferenceRootOccurrenceOf
for myRep in myRef.RepInstances:
    myRepRef = myRep.ReferenceInstanceOf
    if "FEM" == myRepRef.GetAttributeValue("V_discipline"):
        myFemRoot = myRepRef.GetItem("SimFemRoot")
        break

print '*'*40

#Get SIM access for FEM
mySimManagerSIMAccess = myFemRoot.GetItem("SimManagerSIMAccess")
print "root fem:", mySimManagerSIMAccess.ManifestPath

#Get first analysis case
myScenarioManager = mySimuObj.GetItem("SimScenarioManager")
myScenarioAccess = myScenarioManager.AnalysisCases.Item(1).GetItem("SimManagerSIMAccess")
print "analysis case:", myScenarioAccess.ManifestPath

#Get first result
myResultManager = mySimuObj.Results.Item(1).Root.GetItem("SimResultsManager")
myResultAccess = myResultManager.ResultsAnalysisCases.Item(1).GetItem("SimManagerSIMAccess")
print "result case:", myResultAccess.ManifestPath

#Use SIM Reader API to read the results
reader = simResultsReader.SIMResultsReader()
reader.LoadDocument(myResultAccess.ManifestPath)
print "element types:", reader.GetElementSpecs()
...
```

Reading SIM Results Using Python

This article explains how to read results from a SIM file using Python.

This use case can be divided into the following steps:

1. [Import the required modules](#)
2. [Load the SIM document](#)
3. [Get the unit system](#)
4. [Read and print the nodes and node set information](#)
5. [Read and print the elements and element set information](#)
6. [Read and print the steps and frames](#)
7. [Read the field variables](#)
8. [Read the field output data](#)
 1. [Read the orientation information](#)
 2. [Read the drawer](#)
9. [Read the history variables](#)
10. [Read the history output data](#)
11. [Samples](#)

1. Import the required modules

This step imports simResultsReader module which contains methods and classes needed for reading results.

```
...  
from simResultsReader import *  
...
```

2. Load the SIM document

This step loads the SIM document. To automatically upgrade/downgrade the SIM document, refer to 'SIM Utilities reference (python)'

```
...  
reader = SIMResultsReader()  
reader.LoadDocument(simName)  
...
```

3. Get the unit system

This step reads the unit system.

```
...  
unitSys = reader.GetUnitSystem()  
print "Unit System name:", unitSys.name  
print "Unit System conversion factors:", unitSys.convFactors  
print "Length to SI conversion factor", unitSys.lengthToSIConvFactor  
print "Mass to SI conversion factor", unitSys.massToSIConvFactor  
print "Time to SI conversion factor", unitSys.timeToSIConvFactor  
...
```

4. Read and print the nodes and node set information

This step gets the space dimensions of the document, reads the nodes, coordinates and node set information from the SIM document.

```
...  
dims = reader.GetSpaceDimensions()  
print 'Number of space dimensions: %d' % dims  
  
#Read the nodes  
nodes = reader.GetNodeIndices()  
#Read the coordinates  
coords = reader.GetNodeCoordinates(nodes)  
  
print 'Number of nodes: %d\n' % len(nodes)  
counter = 0  
  
for node in nodes:  
    #Read the user label for the node  
    print reader.GetNodeUserLabel(node), ': ',  
    for dim in range(dims):  
        print coords[counter], ' ',  
        counter += 1  
    print ''  
  
#Read the node sets  
nodeSets = reader.GetNodeSets()  
print '\nNumber of node sets: ', len(nodeSets)  
for nodeSet in nodeSets:  
    print ' Node set name: ', nodeSet  
    nodeMembers = reader.GetNodeSetMembers(nodeSet)  
    print '\tMembers: '  
    print '\t', nodeMembers  
...
```

5. Read and print the elements and element set information

This step reads the element specs, elements, element connectivity and element set information from the SIM document.

```
...  
#Read the element specs  
specs = reader.GetElementSpecs()  
  
print '\nNumber of element specs: %d' % len(specs)  
  
for spec in specs:  
    print ' Element spec: ', spec  
    elements = reader.GetElementIndices(spec)  
    #Read the element connectivity  
    connectivity = reader.GetElementsConnectivity(elements)  
    nodesPerElement = len(connectivity)/ len(elements)  
  
    counter = 0  
    for element in elements:  
        print '\t',  
        print reader.GetElementUserLabel(element), ': ',  
        for i in range(nodesPerElement):  
            print connectivity[counter], ' ',  
            counter += 1  
    print ''  
  
#Read the element sets  
elementSets = reader.GetElementSets()  
print '\nNumber of element sets: ', len(elementSets)  
  
for elementSet in elementSets:  
    print ' Element set name: ', elementSet  
    #Read the element set members  
    elementMembers = reader.GetElementSetMembers(elementSet)  
    print '\tMembers: '  
    print '\t', elementMembers  
...
```

6. Read and print the steps and frames

This step reads the steps and frames from the SIM document.

```
...
#Read the step names
stepNames = reader.GetStepSequence()
print '\nNumber of steps: ', len(stepNames)

for stepName in stepNames:
    print ' Step name: ', stepName
    #Read step type and description
    stepInfo = reader.GetStepTypeAndDescription(stepName)
    print ' Step type: %s' % stepInfo[0]
    print ' Step description: %s' % stepInfo[1]

    #Read the frame indices and values
    frameIndices, frameValues = reader.GetFrames(stepName)

    if not frameIndices == None:
        numFrames = len(frameIndices)
        print ' Number of frames: ', numFrames
        for counter in range(numFrames):
            print '\t%d : %g' % (frameIndices[counter], frameValues[counter])
    ...
...
```

7. Read the field variables

This step retrieves the field variables from the SIM document.

```
...
fieldResultsInventory = reader.GetFieldResultsInventory(stepName)
print ' Number of fields: ', len(fieldResultsInventory)

#Sort the list so that the output is consistent
fieldResultsInventory.sort(key = lambda item: item.fieldVariableName)

for field in fieldResultsInventory:
    print ' Field variable: ', field.fieldVariableName
    print '\tField type: ', field.fieldType
    print '\tField index: ', field.fieldIndex
    print '\tField output position: ', field.outputPosition
    print ''
...
```

8. Read the field output data

This step retrieves the field output data from the SIM document. In this example, this step retrieves UNIQUE NODAL data.

```
...
#Initialize a field result
fieldObj = simResultsReader.UniqueNodalFieldReader()
reader.InitFieldResult(stepName, field, fieldObj)
numDrawers = fieldObj.GetNumberOfDrawers()
...

1. Read the orientation information

This step reads the orientation information for this field variable.

...
ori = fieldObj.GetOrientationFieldInfo()
if ori:
    print '\tField has orientation.'
    print '\t Orientation field name: ', ori.fieldVariableName
    print '\t Orientation field type: ', ori.fieldType
    print '\t Orientation field index: ', ori.fieldIndex
    print '\t Orientation field output position: ', ori.outputPosition
    ...

```

2. Read the drawer

This step reads the drawer and prints the field data

```
...
for nd in range(numDrawers):
    print '\t Drawer number: ', nd
    dims = fieldObj.GetDimensions(nd)
    print '\t Frames: ', dims.frameIndices
    print '\t Load cases: ', dims.loadCases
    print '\t Nodes: ', dims.nodeIndices
    print '\t Components: ', dims.components
    print '\t Invariants: ', dims.invariants

    frames = dims.frameIndices
    lcs = dims.loadCases
    nodes = dims.nodeIndices
    for frame in frames:
        print '\n\t Processing frame: ', frame
        for lc in lcs:
            print '\t Processing load case: ', lc
            data, imagData = fieldObj.GetFieldResultData(nd, frame, nodes, lc)
            print '\t Real data: ', data
            print '\t Imag data: ', imagData
    ...

```

9. Read the history variables

This step retrieves the history variables from the SIM document.

```
...
historyResultsInventory = reader.GetHistoryResultsInventory(stepName)
print ' Number of history: ', len(historyResultsInventory)

#Sort the list so that the output is consistent
historyResultsInventory.sort(key = lambda item: item.fieldVariableName)

for history in historyResultsInventory:
    print ' Processing history output: ', history.fieldVariableName
    print '\tHistory output type: ', history.fieldType
    print '\t\tHistory output index: ', history.fieldIndex
    print '\t\tHistory output position: ', history.outputPosition
    print ''
```

10. Read the history output data

This step retrieves the history output data from the SIM document. In this example, this step retrieves UNIQUE NODAL data, reads the drawer and prints the history data.

```
...
#Initialize a history result
historyObj = simResultsReader.UniqueNodalFieldReader()
reader.InitHistoryResult(stepName, field, historyObj)
numDrawers = historyObj.GetNumberOfDrawers()

for nd in range(numDrawers):
    print '\t Processing drawer: ', nd
    dims = historyObj.GetDimensions(nd)
    print '\t\t Load cases: ', dims.loadCases
    print '\t\t Nodes: ', dims.nodeIndices
    print '\t\t Components: ', dims.components
    print '\t\t Invariants: ', dims.invariants

    frames = dims.frameIndices
    lcs = dims.loadCases
    nodes = dims.nodeIndices
    for node in nodes:
        print '\n\t\t Processing node: ', node
        for lc in lcs:
            print '\t\t\t Processing load case: ', lc
            for component in dims.components:
                print '\t\t\t\t Processing component: ', component
                data, imagData = historyObj.GetHistoryResultData(nd, node, component, lc)
                print '\t\t\t\t Real data: ', data
                print '\t\t\t\t Imag data: ', imagData
...

```

11. Samples

Read results from a SIM document: [simResultsReaderSample1.py](#)

Read results from a SIM document which has Unicode strings: [simResultsReaderSample1-Unicode.py](#)

Read mises stress values from a SIM document: [misesFromSim.py](#)

Read and write results to the same SIM document: [resultsReadWriteSample1.py](#)

Copyright © 1999-2018, Dassault Systèmes. All rights reserved.

Writing SIM Results Using Python

This article explains how to write results to a SIM file using Python.

This use case can be divided into the following steps:

1. [Import the required modules](#)
2. [Create a new SIM document](#)
3. [Set the unit system](#)
4. [Write the nodes and node set information](#)
5. [Write the elements and element set information](#)
6. [Write the material and section information](#)
7. [Create a new step and frames](#)
8. [Create and initialize a new nodal displacement field variable](#)
9. [Create a new field type and write data](#)
10. [Create a new history type and write history data](#)
11. [Save and close the document](#)
12. [Samples](#)
13. [Adding user labels](#)
14. [SIM Predefined string](#)

1. Import the required modules

This step imports the simResultsWriter module, which contains methods and classes needed for writing results.

```
...
from simResultsWriter import *
...
```

2. Create a new SIM document

This step creates a new SIM document.

```
iDocName = "new.sim"
writer = SIMResultsWriter()
writer.Initialize()
writer.CreateDocument(iDocName)
...
```

3. Set the unit system

This step sets the unit system for the SIM document.

```
...
#Set the unit system
iUnitSystemName = "FPSUnitSystem"
iLengthToSICconvFactor = 0.3048
iMassToSICconvFactor = 0.453592
iTimeToSICconvFactor = 1.0
writer.SetUnitSystem(iUnitSystemName, iLengthToSICconvFactor, iMassToSICconvFactor, iTimeToSICconvFactor)
...
```

4. Write the nodes and node set information

This step writes the nodes, coordinates and node set information to the SIM document. The user is in charge of the management of the memory. The user has to a

```
...
#Write nodes
nodeIndices = [ 0,1,2,3,4,5,6,7, 8, 9, 10, 11, 12 ]
coords = [ 0.0, 0.0 , 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0,
numOfSpaceDims = 3
writer.WriteNodes(numOfSpaceDims, nodeIndices, coords)

#Create node set
allNodes = "ALL NODES"
setIndices = [ 0,1,2,3,4,5,6,7 ]
writer.CreatePermutedNodeSet(allNodes, setIndices)
...
```

5. Write the elements and element set information

This step writes the element specs, elements, element connectivity and element set information to the SIM document.

```
...
#Write elements
elmType = "C3D4";
elementIndices = [ 0,1,2,3 ];
numElements = len(elementIndices)
connectivity = [ 0, 1, 2, 5, 0, 2, 3, 7, 0, 5, 7, 4, 5, 0, 7, 2 ]
writer.AddElements(elmType, elementIndices, connectivity)

#Create element set
setIndices = [ 0, 1 ]
aluminiumElements = "ALUMINIUM_ELEMENTS"
writer.CreatePermutedElementSet(aluminiumElements, setIndices)
...
```

6. Write the material and section information

This step writes the material and section information to the SIM document and assigns them to the elements sets created in previous step.

```
...
#create material
materialName = "ALUMINIUM"
writer.CreateMaterial(materialName)

#create section
writer.CreateContinuumSection(aluminiumElements, materialName)
...
```

7. Create a new step and frames

This step creates a new static step and frames.

```
...
frameValues = [ 0.0, 1.0]

stepName = "Step-1"
stepType = "Step_Gen_Static"
stepDescription = "The description goes here"

writer.AddStep(stepName, stepType, stepDescription, frameValues)
...
```

8. Create and initialize a new nodal displacement field variable

This step creates and initializes a nodal displacement field variable. The dimension of the field is also defined here (in this case, it is a vector with 3 components).

```
...
#Initialize the field variable
dispUniqueNodalField = UniqueNodalField()
stepName = "Step-1"
displacementFieldType = "U_D"
writer.InitFieldResult(stepName, displacementFieldType, dispUniqueNodalField)

#define dimensions
vectorComponents = ["C_1", "C_2", "C_3"]
magInvariants = ["MAGNITUDE"]
complexData = False
doublePrecision = True
dispUniqueNodalField.DefineDimensions(allNodes, vectorComponents, magInvariants, complexData, doublePrecision)
...
```

9. Create a new field type and write field data

This step creates a new displacement field type and writes the data at the first frame.

10. Create a new history type and write history data

This step creates a new displacement history type and writes the data at all frames.

```

...
#Initialize the history variable
dispHistoryField = UniqueNodalField()
writer.InitHistoryResult(stepName, displacementFieldType, dispHistoryField)
vectorComponents = ["C_1"]
invariants = None
dispHistoryField.DefineDimensions(allNodes, vectorComponents, invariants, complexData, doublePrecision)

#Write the history data
dispHistoryData = [ 0,0,0,0,0,0,-1.52,-1.25,-9.43,-5.95,-2.25,1.54,5.27,8.80 ]
frameIndices = [ 0,1 ]
dispHistoryField.AddHistoryData(frameIndices, allNodes, dispHistoryData)
...

```

11. Close the document

This step saves and closes the document.

```
...  
writer.Finalize()  
...
```

12. Samples

Write results to a SIM document #1: [simResultsWriterSample1.py](#)

Write results to a SIM document #2: [simResultsWriterSample2.py](#)

Write user node and element labels to a SIM document: [simResultsWriterSample3.py](#)

Write results to a SIM document with some Unicode strings: [simResultsWriterSample1-Unicode.py](#)

Read and write results to the same SIM document: [resultsReadWriteSample1.py](#)

13. Adding user labels

This step shows how to add user node labels to the SIM document, map the user labels to the flat labels and how the user labels are used. For a sample script, please see [here](#).

`AddUserNodeLabels()` is used to add user node labels to the SIM document:

```
...
#Add the user node labels
instance1 = "Part-1-1"
userNodeLabels = [1001,1002,1003,1004,1005,1006,1007,1008]
flatNodeIndices = writer.AddUserNodeLabels(iInstanceName = instance1, iUserLabels = userNodeLabels)
```

A dictionary is constructed to store the map between the user supplied node labels and the flat labels generated by SIM:

```
...
nodeMap = {}
for i in range(len(userNodeLabels)):
    nodeMap[instance1 + "." + str(userNodeLabels[i])] = flatNodeIndices[i]
```

When creating the element connectivity or node sets, the above dictionary is used:

```

...
elementSpec = "C3D4";
elementIndices = [1,2]
connectivity = [nodeMap[instance1+"."+str(1005)], nodeMap[instance1+"."+str(1006)], nodeMap[instance1+"."+str(1008)], ...]
writer.AddElements(elementSpec, elementIndices, connectivity)

nset1 = "SampleNodeSet1"
nset1Indices = [nodeMap[instance1+"."+str(1006)], nodeMap[instance1+"."+str(1008)]]
writer.CreatePermutedNodeSet(nset1, nset1Indices)
...

```

14. SIM predefined string

This section discusses some of the possible predefined SIM strings that can be used in the API.

- Step Type

Predefined String	Description
Step_Gen_Anneal	*ANNEAL
Step_Gen_BaseStateResolution	Base state step for linear perturbation steps when there is no general step preceding them

Step_Gen_TemperatureDisplacement_SteadyState	*COUPLED TEMPERATURE-DISPLACEMENT, STEADY STATE
Step_Gen_TemperatureDisplacement_Transient	*COUPLED TEMPERATURE-DISPLACEMENT, TRANSIENT
Step_Gen_TemperatureElectricalSteadyState	*COUPLED THERMAL-ELECTRICAL, STEADY STATE
Step_Gen_TemperatureElectrical	COUPLED THERMAL-ELECTRICAL
Step_Gen_DirectCyclic	*DIRECT CYCLIC
Step_Gen_Dynamic	*DYNAMIC
Step_Gen_Dynamic_Explicit	*DYNAMIC, EXPLICIT
Step_Gen_Dynamic_Explicit_TemperatureDisplacement	*DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT
Step_Gen_Geostatic	*GEOSTATIC
Step_Gen_HeatTransferSteadyState	*HEAT TRANSFER, STEADY STATE
Step_Gen_HeatTransfer	*HEAT TRANSFER
Step_Gen_Dynamic_Implicit	*DYNAMIC
Step_Gen_MassDiffusionSteadyState	*MASS DIFFUSION, STEADY STATE
Step_Gen_MassDiffusion	*MASS DIFFUSION
Step_Gen_SoilsSteadyState	*SOILS
Step_Gen_Soils	*SOILS, CONSOLIDATION
Step_Gen_StaticRiks	*STATIC, RIKS
Step_Gen_Static	*STATIC
Step_Gen_SteadyStateTransport	*STEADY STATE TRANSPORT
Step_Gen_Dynamic_SubspaceProjection	*DYNAMIC, SUBSPACE
Step_Gen_Visco	*VISCO
Step_Gen_Electromagnetic	*ELECTROMAGNETIC
Step_Lin_Buckle	*STEP, LINEAR and *BUCKLE
Step_Lin_ComplexFrequency	*STEP, LINEAR and *COMPLEX FREQUENCY
Step_Lin_Frequency	*STEP, LINEAR and *FREQUENCY
Step_Lin_MatrixGeneration	*STEP, LINEAR and *MATRIX GENERATE
Step_Lin_SubstructureGeneration	*STEP, LINEAR and *SUBSTRUCTURE GENERATE
Step_Lin_BeamSectionGeneration	*STEP, LINEAR and *BEAM SECTION GENERATE
Step_Lin_DirectHarmonic	*STEP, LINEAR and *STEADY STATE DYNAMICS, DIRECT
Step_Lin_ModalHarmonic	*STEP, LINEAR and *STEADY STATE DYNAMICS
Step_Lin_ModalRandomResponse	*STEP, LINEAR and *RANDOM RESPONSE
Step_Lin_ModalTransient	*STEP, LINEAR and *MODAL DYNAMIC
Step_Lin_ResponseSpectrum	*STEP, LINEAR and *RESPONSE SPECTRUM
Step_Lin_Static	*STEP, LINEAR and *STATIC
Step_Lin_Electromagnetic	*STEP, LINEAR and *ELECTROMAGNETIC

- Component

Predefined String	Description
C_SCALAR	Indicates scalar
C_1	Indicates vector first component
C_2	Indicates vector second component
C_3	Indicates vector third component
C_11	Indicates second-order tensor 11-component
C_22	Indicates second-order tensor 22-component
C_33	Indicates second-order tensor 33-component
C_12	Indicates second-order tensor 12-component
C_13	Indicates second-order tensor 13-component
C_23	Indicates second-order tensor 23-component
C_21	Indicates second-order tensor 21-component
C_31	Indicates second-order tensor 31-component
C_32	Indicates second-order tensor 32-component
C_12PLUS	Indicates second-order tensor 12-component PLUS cross-diagonal 21 component (and, when used with a symmetric tensor, therefore TW 'engineering' 12 shear component).
C_13PLUS	Indicates second-order tensor 13-component PLUS cross-diagonal 31 component.
C_23PLUS	Indicates second-order tensor 23-component PLUS cross-diagonal component.
C_SORT_PRIN1	Indicates second-order tensor principal value 1, as sorted with principal values 1,2,3 ascending.
C_SORT_PRIN2	Indicates second-order tensor principal value 2, as sorted with principal values 1,2,3 ascending.
C_SORT_PRIN3	Indicates second-order tensor principal value 3, as sorted with principal values 1,2,3 ascending.
C_1111	Indicates fourth-order tensor 1111-component
C_1122	Indicates fourth-order tensor 1122-component
C_1112	Indicates fourth-order tensor 1112-component
C_2211	Indicates fourth-order tensor 2211-component
C_2222	Indicates fourth-order tensor 2222-component
C_2212	Indicates fourth-order tensor 2212-component
C_1211	Indicates fourth-order tensor 1211-component
C_1222	Indicates fourth-order tensor 1222-component
C_1212	Indicates fourth-order tensor 1212-component
C_ROT1	Indicates the 1-component of a finite rotation vector, representing an orthogonal second-order tensor.
C_ROT2	Indicates the 2-component of a finite rotation vector, representing an orthogonal second-order tensor.

C_ROT	Indicates the 3-component of a finite rotation vector, representing an orthogonal second-order tensor.
C_PRIN1	Indicates second-order tensor principal value 1. Differs from C_SORT_PRIN1 in that the 1 here reflects a basis vector 1, not the lowest c in conjunction with a quaternion to represent a symmetric tensor; or with two quaternions to represent a general tensor.
C_PRIN2	See C_PRIN1
C_PRIN3	See C_PRIN1

C_QUAT0 Indicates quaternion component of orthogonal two-tensor. Note: rotation in three dimensions has all four quaternion components possibly involves nonzero components only for C_QUAT0 and C_QUAT3.

C_QUAT1	Indicates quaternion component of orthogonal two-tensor; component 1 of vector q.
C_QUAT2	Indicates quaternion component of orthogonal two-tensor; component 2 of vector q.
C_QUAT3	Indicates quaternion component of orthogonal two-tensor; component 3 of vector q.

- **Invariant**

Predefined String	Description
INV1	The 'first invariant' of a second order tensor. Applied to a strain, this gives volumetric strain
INV3	The 'third invariant' of a second order tensor.
PRESSURE_LIKE	This is defined as minus one third INV1. Applied to stress tensor, it produces pressure.
MISES_LIKE	The function which produces the Mises stress from a second-order stress tensor.
TRESCA_LIKE	The function which produces the Tresca stress from a second-order stress tensor.
MAGNITUDE	The function which produces the magnitude of a vector.
MAX_PRINCIPAL	The function which produces the maximum principal value of a second-order tensor.
MID_PRINCIPAL	The function which produces the 'mid' principal value of a second-order tensor.
MIN_PRINCIPAL	The function which produces the minimum principal value of a second-order tensor.
MAX_INPLANE_PRINCIPAL	The function which produces the maximum in-plane principal value of a second-order tensor.
MIN_INPLANE_PRINCIPAL	The function which produces the minimum in-plane principal value of a second-order tensor.
OUTOFPANE_PRINCIPAL	The function which produces the out of plane principal value of a second-order tensor.

- **Output Position**

Predefined String	Description
UNIQUE_NODAL	UNIQUE NODAL
ELEMENT_CENTROID	ELEMENT CENTROID
WHOLE_ELEMENT	WHOLE ELEMENT
INTEGRATION_POINT	INTEGRATION POINT
ELEMENT_NODAL	ELEMENT NODAL
SURFACE_NODAL	SURFACE NODAL
WHOLE_REGION	WHOLE REGION
WHOLE_MODEL	WHOLE MODEL
GENERIC_POSITION	GENERIC POSITION

- **Element type** (please refer to Abaqus Element Guide for the complete list)

Predefined String	Description
C3D8	C3D8
C3D4	C3D4
B31	B31
S4	S4
...	...

- **Physical dimension**

Physical dimension describes the dimensions of a quantity in terms of fundamental quantities. This is used for units conversion. The following are the rules to define the physical dimension string:

1. Write its formula in the base physical dimensions, representing the base physical dimensions (M..L..T..E..K..A..C) through their own symbol names.
2. Omit any exponents of value 1 in the formula; include no base dimensions which have an exponent 0 in the formula.
3. Arrange the formula so the base physical dimensions appear in the order they appear below, that is, M..L..T..E..K..A..C.
4. Write down what you see from left to right, each base symbol followed immediately by what we'll call the symbol form of its exponent:
5. To put an exponent in symbol form, as a fraction it must be normalized; represent minus in the numerator with a prefix 0.)

Predefined String	Description
L	Length
M	Mass
T	Time
E	Electric Current
K	Temperature
A	Amount of substance
C	Luminous Intensity
MLT02	Mass*Length/Time^2; physical dimension for stress
ML2T02K01A01	Mass*Length^2/(Time^2*Temperature*Amount of substance)
M1_2L01_2T01	Mass^0.5/(Length^0.5*Time)
...	...

- **Field type**

Use GetFieldTypeFromOutputVariable() to get a field type from Abaqus output variable (refer to Abaqus Output Guide for the complete list of Abaqus output variables). For user defined field types, use either AddFieldType() or AddModifiedFieldType() to create new field types.

Predefined String	Description
U_D	Translational displacement
STRAIN_ENERGY	Strain energy

S0	Stress
...	...

SIMResultsReader

This class is intended to make it as easy as possible to read results from SIM.

Method Index

- o [init](#)
Creates a new instance of SIMResultsReader
- o [Finalize](#)
Delete all the objects created.
- o [GetBaseStateStep](#)
Gets the base state step name.
- o [GetElementFaceConnectivity](#)
Gets the face connectivity of the given elements at the given face.
- o [GetElementIndices](#)
Gets the element indices for the specified element spec.
- o [GetElementSetMembers](#)
Gets the element indices in the specified element set.
- o [GetElementSetName](#)
Gets the user element set name.
- o [GetElementSets](#)
Gets the set of element sets in the model.
- o [GetElementSpec](#)
Gets the element spec of the given element.
- o [GetElementSpecs](#)
Gets the vector of element specs.
- o [GetElementTopology](#)
Gets the element topology of the given element.
- o [GetElementUserLabel](#)
Gets the user label for an element. Returns true if user label found.
- o [GetElementsConnectivity](#)
Gets the element connectivity for the specified element indices. The element indices must belong to the same element spec.
- o [GetElementsInSection](#)
Gets the elements in the given section.. If iElemIndices is specified, gets the elements that belong to the given section from this list of elements .
- o [GetFieldResultsInventory](#)
Gets all the field info objects describing the field types and output locations for the field results in the step.
- o [GetFrames](#)
Gets the frame indices and the frame values for the specified step.
- o [GetHistoryResultsInventory](#)
Gets all the field info objects describing the field types and output locations for the history results in the step.
- o [GetLoadCases](#)
Gets the map of load cases in the model.
- o [GetNodeCoordinates](#)
Gets the nodal coordinates.
- o [GetNodeIndices](#)
Gets the node indices.
- o [GetNodeSetMembers](#)
Gets the node indices in the specified node set.
- o [GetNodeSetName](#)
Gets the user node set name.
- o [GetNodeSets](#)
Gets the set of node sets in the model.
- o [GetNodeUserLabel](#)
Gets the user label for a node. Returns true if user label found.
- o [GetSectionNames](#)
Gets the name of the sections present in the model.

- o [GetSectionType](#)
Gets the section type for the specified section.
- o [GetShellSectionProperties](#)
Gets the shell section properties for the given section.
- o [GetSpaceDimensions](#)
Returns the space dimensions of the main model in the SIM document.
- o [GetStepSequence](#)
Gets the sequence of steps.
- o [GetStepTypeAndDescription](#)
Gets the step type and the step description for the specified step.
- o [GetSurfaceElementsAtFace](#)
Gets the elements at the given face in the surface.
- o [GetSurfaceFaces](#)
Gets the faces in the given surface.
- o [GetSurfaceNames](#)
Gets the name of surfaces in the model.
- o [GetUnitSystem](#)
Returns the unit system name of the SIM document and the conversion factors.
- o [GroupElementsBySections](#)
Groups the elements based on its section.
- o [HasExternalElementSets](#)
Checks if external element sets exists in the sim file. Returns true if external element sets are found.
- o [HasExternalNodeSets](#)
Checks if external node sets exists in the sim file. Returns true if external node sets are found.
- o [HasInternalElementSets](#)
Checks if internal element sets exists in the sim file. Returns true if internal element sets are found.
- o [HasInternalNodeSets](#)
Checks if internal node sets exists in the sim file. Returns true if internal node sets are found.
- o [InitFieldResult](#)
Initializes a field for reading field result.
- o [InitHistoryResult](#)
Initializes a field for reading history result.
- o [LoadDocument](#)
Loads an existing SIM document.

Methods

- o [__init__](#)
Parameters:
None
- o [Finalize](#)
Delete all the objects created.
Parameters:
None
- o [GetBaseStateStep](#)
Gets the base state step name.
Parameters:
[out] A string specifying the base state step name.
- o [GetElementFaceConnectivity](#)
Gets the face connectivity of the given elements at the given face.
Parameters:
iElementIndices
[in/required] The element indices.
iFaceIndex
[in/required] The face index.
[out] The face connectivity of the given elements at the given face.
- o [GetElementIndices](#)
Gets the element indices for the specified element spec.
Parameters:
iElementSpec
[in/required] A string specifying the element spec.
[out] A sequence of integers specifying the element indices.
- o [GetElementSetMembers](#)

Gets the element indices in the specified element set.

Parameters:

iElementSet
[in|required] A string specifying the element set name.
[out] A sequence of integers specifying the elements in the element set; if the set is unsorted, the members will be unsorted.

o **GetElementSetName**

Gets the user element set name.

Parameters:

iElementSet
[in|required] A string specifying the flat element set name.
[out] A sequence of strings specifying the occurrence name and the user element set name.

o **GetElementSets**

Gets the set of element sets in the model.

Parameters:

includeExternalSets
[in/optional] A boolean specifying if external sets should be included or not.
includeInternalSets
[in/optional] A boolean specifying if internal sets should be included or not.
[out] A sequence of strings specifying the element sets in the model.

o **GetElementSpec**

Gets the element spec of the given element.

Parameters:

iElementIndex
[in|required] The element index.
[out] The element spec of the given element.

o **GetElementSpecs**

Gets the vector of element specs.

Parameters:

[out] A sequence of strings specifying the element specs in the model.

o **GetElementTopology**

Gets the element topology of the given element.

Parameters:

iElementIndex
[in|required] The element index.
[out] The element topology of the given element.

o **GetElementUserLabel**

Gets the user label for an element. Returns true if user label found.

Parameters:

iElementIndex
[in|required] An integer specifying the element for which user label is required.
[out] A tuple of the string specifying the occurrence name and the integer specifying the element user label.

o **GetElementsConnectivity**

Gets the element connectivity for the specified element indices. The element indices must belong to the same element spec.

Parameters:

iElemIndices
[in|required] A sequence of integers specifying the element indices.
[out] A sequence of integers specifying the element connectivity.

o **GetElementsInSection**

Gets the elements in the given section.. If iElemIndices is specified, gets the elements that belong to the given section from this list of elements .

Parameters:

iSectionName
[in|required] The name of the section.
iElementIndices
[in/optional] The list of element indices.
[out] The elements in the section.

o **GetFieldResultsInventory**

Gets all the field info objects describing the field types and output locations for the field results in the step.

Parameters:

iStepName
[in|required] A string specifying the step name.
[out] A sequence of FieldInfo objects specifying the field info sequence.

o **GetFrames**

Gets the frame indices and the frame values for the specified step.

Parameters:

iStepName
[in|required] A string specifying the step name.
[out] A sequence of sequence of integers specifying the frame indices and a sequence of doubles specifying the frame values.

o **GetHistoryResultsInventory**

Gets all the field info objects describing the field types and output locations for the history results in the step.

Parameters:

iStepName

[in|required] A string specifying the step name.
[out] A sequence of FieldInfo objects specifying the field info sequence.

o GetLoadCases

Gets the map of load cases in the model.

Parameters:
[out] A dictionary with the load case index as keys and the load case name as values.

o GetNodeCoordinates

Gets the nodal coordinates.

Parameters:
iNodeIndices
[in|required] A sequence of integers specifying the node indices.
[out] A sequence of doubles specifying the nodal coordinates.

o GetNodeIndices

Gets the node indices.

Parameters:
[out] A sequence of integers specifying the node indices.

o GetNodeSetMembers

Gets the node indices in the specified node set.

Parameters:
iNodeSet
[in|required] A string specifying the node set name.
[out] A sequence of integers specifying the nodes in the node set; if the set is unsorted, the members will be unsorted.

o GetNodeSetName

Gets the user node set name.

Parameters:
iNodeSet
[in|required] A string specifying the flat node set name.
[out] A sequence of a strings specifying the occurrence name and the user node set name.

o GetNodeSets

Gets the set of node sets in the model.

Parameters:
includeExternalSets
[in|optional] A boolean specifying if external sets should be included or not.
includeInternalSets
[in|optional] A boolean specifying if internal sets should be included or not.
[out] A sequence of strings specifying the node sets in the model.

o GetNodeUserLabel

Gets the user label for a node. Returns true if user label found.

Parameters:
iNodeIndex
[in|required] An integer specifying the node number for which user label is required.
[out] A sequence of a string specifying the occurrence name and an integer specifying the node user label

o GetSectionNames

Gets the name of the sections present in the model.

Parameters:
[out] The name of the sections in the model.

o GetSectionType

Gets the section type for the specified section.

Parameters:
iSectionName
[in|required] The name of the section.
iSectionType
[in|required] The type of the section.

o GetShellSectionProperties

Gets the shell section properties for the given section.

Parameters:
iSectionName
[in|required] The name of the section.
[out] A list containing the section integration rule, number of layers in the section, angle, thickness, integration points in each layer. *

o GetSpaceDimensions

Returns the space dimensions of the main model in the SIM document.

Parameters:
None

o GetStepSequence

Gets the sequence of steps.

Parameters:
[out] A sequence of strings specifying the steps in the model.

o GetStepTypeAndDescription

Gets the step type and the step description for the specified step.

Parameters:

iStepName
[in|required] A string specifying the step name.
[out] A sequence of a string specifying the step type and a string specifying the step description.

o GetSurfaceElementsAtFace

Gets the elements at the given face in the surface.

Parameters:

iSurfaceName
[in|required] A string specifying the surface name.
iFaceIndex
[in|required] An integer specifying the face index.
[out] A sequence of integers specifying the elements at the face in the surface..

o GetSurfaceFaces

Gets the faces in the given surface.

Parameters:

iSurfaceName
[in|required] A string specifying the surface name.
[out] A sequence of integers specifying the faces in the surface.

o GetSurfaceNames

Gets the name of surfaces in the model.

Parameters:

[out] A sequence of strings specifying the surfaces in the model.

o GetUnitSystem

Returns the unit system name of the SIM document and the conversion factors.

Parameters:

[out] An instance of the UnitSystem class object.

o GroupElementsBySections

Groups the elements based on its section.

Parameters:

iElemIndices
[in|required] The element indices.
sectionElemIndicesMap
[in|required] The elements from the input set of elements grouped based on their sections.

o HasExternalElementSets

Checks if external element sets exists in the sim file. Returns true if external element sets are found.

Parameters:

[out] True if external element sets exists in the sim file.

o HasExternalNodeSets

Checks if external node sets exists in the sim file. Returns true if external node sets are found.

Parameters:

[out] True if external node sets exists in the sim file.

o HasInternalElementSets

Checks if internal element sets exists in the sim file. Returns true if internal element sets are found.

Parameters:

[out] True if internal element sets exists in the sim file.

o HasInternalNodeSets

Checks if internal node sets exists in the sim file. Returns true if internal node sets are found.

Parameters:

[out] True if internal node sets exists in the sim file.

o InitFieldResult

Initializes a field for reading field result.

Parameters:

iStepName
[in|required] A string specifying the step name for the result.
iFieldInfo
[in|required] A FieldInfo object specifying the field info object.
ioFieldReader
[in|required] A FieldReader object specifying the field object which should be used to read data.

o InitHistoryResult

Initializes a field for reading history result.

Parameters:

iStepName
[in|required] A string specifying the step name for the result.
iFieldInfo
[in|required] A FieldInfo object specifying the field info object.
ioFieldReader
[in|required] A FieldReader object specifying the field object which should be used to read data.

o LoadDocument

Loads an existing SIM document.

Parameters:

iDocument

[in|required] A string specifying the document name or a SIMDocument object specifying the SIMDocument object.

ElementCentroidFieldReader

The ElementCentroidFieldReader object is derived from the FieldReader object.

Method Index

o __init__

Creates a new instance of ElementCentroidFieldReader object

o GetDimensions

Returns an instance of ElementCentroidDimension for the drawer with members: frame indices, loadcase indices, section points, element indices, Component labels, Invariants, isDoublePrecision, isComplex

o GetFieldResultData

Returns a tuple of real and imaginary data for the drawer.

o GetHistoryResultData

Returns a tuple of real and imaginary data for the drawer.

o GetOrientationFieldInfo

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Methods

o __init__

Creates a new instance of ElementCentroidFieldReader object

Parameters:

[out] A new instance of ElementCentroidFieldReader object

o GetDimensions

Returns an instance of ElementCentroidDimension for the drawer with members: frame indices, loadcase indices, section points, element indices, Component labels, Invariants, isDoublePrecision, isComplex

Parameters:

iDrawerNumber

[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

[out] An instance of the ElementCentroidDimension object with members" frame indices, loadCases, section points, element indices, Component labels, Invariants, isDoublePrecision, isComplex.

o GetFieldResultData

Returns a tuple of real and imaginary data for the drawer.

Parameters:

iDrawerNumber

[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

iFrameIndex

[in|required] An integer specifying the frame index for which the data is requested; Set it to -1 if the field does not have associated frames.

iSectionPoint

[in|required] An integer specifying the section point index for which data is required.

iElementIndices

[in|required] A sequence of integers specifying the element indices for which data is required.

iLoadCase

[in|optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is - 1.

[out] A sequence of doubles specifying the real and imaginary data for the drawer.

o GetHistoryResultData

Returns a tuple of real and imaginary data for the drawer.

Parameters:

iDrawerNumber

[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

iSectionPoint

[in|required] An integer specifying the section point index for which data is required.

iElementIndex

[in|required] An integer specifying the element index for which data is required.

iComponentSymbol

[in|required] A string specifying the component label for which data is required.

iLoadCase

[in|optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is - 1.

[out] A sequence of doubles specifying the real and imaginary data for the drawer.

o GetOrientationFieldInfo

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Parameters:

[out] A FieldInfo object specifying the orientation field info.

ElementFaceFieldReader

The ElementFaceFieldReader object is derived from the FieldReader object.

Method Index

- o [__init__](#)
Creates a new instance of ElementFaceField object
- o [GetDimensions](#)
Returns an instance of ElementIntPointDimension for the drawer with members: frame indices, loadcase indices, section points, element indices, face ID, component labels, Invariants, isDoublePrecision, isComplex
- o [GetFieldResultData](#)
Returns a tuple of real and imaginary data for the drawer.
- o [GetHistoryResultData](#)
Returns a tuple of real and imaginary data for the drawer.
- o [GetOrientationFieldInfo](#)
Gets the associated orientation field info. Returns false if the field has no associated orientation.

Methods

- o [__init__](#)
Creates a new instance of ElementFaceField object
Parameters:
[out] A new instance of ElementFaceField object
- o [GetDimensions](#)
Returns an instance of ElementIntPointDimension for the drawer with members: frame indices, loadcase indices, section points, element indices, face ID, component labels, Invariants, isDoublePrecision, isComplex
Parameters:
iDrawerNumber
[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
[out] An instance of the ElementIntPointDimension object with members: frame indices, loadcase indices, section points, element indices, face ID, component labels, Invariants, isDoublePrecision, isComplex.
- o [GetFieldResultData](#)
Returns a tuple of real and imaginary data for the drawer.
Parameters:
iDrawerNumber
[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
iFrameIndex
[in|required] An integer specifying the frame index for which the data is requested; Set it to -1 if the field does not have associated frames.
iSectionPoint
[in|required] An integer specifying the section point index for which data is required.
iFaceID
[in|required] An integer specifying the face ID for which data is required.
iElementIndices
[in|required] A sequence of integers specifying the element indices for which data is required.
iLoadCase
[in|optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.
[out] A sequence of doubles specifying the real and imaginary data for the drawer.
- o [GetHistoryResultData](#)
Returns a tuple of real and imaginary data for the drawer.
Parameters:
iDrawerNumber
[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
iSectionPoint
[in|required] An integer specifying the section point index for which data is required.
iElementIndex
[in|required] An integer specifying the element index for which data is required.
iFaceID
[in|required] An integer specifying the face ID for which data is required.
iComponentSymbol
[in|required] A string specifying the component label for which data is required.
iLoadCase
[in|optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.
[out] A sequence of doubles specifying the real and imaginary data for the drawer.
- o [GetOrientationFieldInfo](#)
Gets the associated orientation field info. Returns false if the field has no associated orientation.
Parameters:
[out] A FieldInfo object specifying the orientation field info.

ElementIntegrationPointFieldReader

The ElementIntegrationPointFieldReader object is derived from the FieldReader object.

Method Index

- o [__init__](#)
Creates a new instance of ElementIntegrationPointField object
- o [GetDimensions](#)
Returns an instance of ElementIntPointDimension for the drawer with members: frame indices, loadcase indices, section points, element indices, integration point indices, component labels, Invariants, isDoublePrecision, isComplex
- o [GetFieldResultData](#)
Returns a tuple of real and imaginary data for the drawer.
- o [GetHistoryResultData](#)
Returns a tuple of real and imaginary data for the drawer.
- o [GetOrientationFieldInfo](#)
Gets the associated orientation field info. Returns false if the field has no associated orientation.

Methods

- o [__init__](#)
Creates a new instance of ElementIntegrationPointField object
Parameters:
[out] A new instance of ElementIntegrationPointField object
- o [GetDimensions](#)
Returns an instance of ElementIntPointDimension for the drawer with members: frame indices, loadcase indices, section points, element indices, integration point indices, component labels, Invariants, isDoublePrecision, isComplex
Parameters:
iDrawerNumber
[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
[out] An instance of the ElementIntPointDimension object with members: frame indices, loadcase indices, section points, element indices, integration point indices, component labels, Invariants, isDoublePrecision, isComplex.
- o [GetFieldResultData](#)
Returns a tuple of real and imaginary data for the drawer.
Parameters:
iDrawerNumber
[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
iFrameIndex
[in|required] An integer specifying the frame index for which the data is requested; Set it to -1 if the field does not have associated frames.
iSectionPoint
[in|required] An integer specifying the section point index for which data is required.
iElementIndices
[in|required] A sequence of integers specifying the element indices for which data is required.
iLoadCase
[in|optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.
[out] A sequence of doubles specifying the real and imaginary data for the drawer.
- o [GetHistoryResultData](#)
Returns a tuple of real and imaginary data for the drawer.
Parameters:
iDrawerNumber
[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
iSectionPoint
[in|required] An integer specifying the section point index for which data is required.
iElementIndex
[in|required] An integer specifying the element index for which data is required.
iIntegrationPoint
[in|required] An integer specifying the integration point index for which data is required.
iComponentSymbol
[in|required] A string specifying the component label for which data is required.
iLoadCase
[in|optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.
[out] A sequence of doubles specifying the real and imaginary data for the drawer.
- o [GetOrientationFieldInfo](#)
Gets the associated orientation field info. Returns false if the field has no associated orientation.
Parameters:
[out] A FieldInfo object specifying the orientation field info.

ElementNodalFieldReader

The ElementNodalFieldReader object is derived from the FieldReader object.

Method Index

- o [__init__](#)
Creates a new instance of ElementNodalFieldReader object

- o [GetDimensions](#)

Returns an instance of ElementNodalDimension for the drawer with members: frame indices, loadcase indices, section points, element indices, local node indices, component labels, Invariants, isDoublePrecision, isComplex of data per element class for element output or all the nodes for node output.

- o [GetFieldResultData](#)

Returns a tuple of real and imaginary data for the drawer.

- o [GetHistoryResultData](#)

Returns a tuple of real and imaginary data for the drawer.

- o [GetOrientationFieldInfo](#)

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Methods

- o [__init__](#)

Creates a new instance of ElementNodalFieldReader object

Parameters:

[out] A new instance of ElementNodalFieldReader object

- o [GetDimensions](#)

Returns an instance of ElementNodalDimension for the drawer with members: frame indices, loadcase indices, section points, element indices, local node indices, component labels, Invariants, isDoublePrecision, isComplex of data per element class for element output or all the nodes for node output.

Parameters:

iDrawerNumber

[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

[out] An instance of the ElementNodalDimension object with members: frame indices, loadcase indices, section points, element indices, local node indices, component labels, Invariants, isDoublePrecision, isComplex.

- o [GetFieldResultData](#)

Returns a tuple of real and imaginary data for the drawer.

Parameters:

iDrawerNumber

[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

iFrameIndex

[in|required] An integer specifying the frame index for which the data is requested; Set it to -1 if the field does not have associated frames.

iSectionPoint

[in|required] An integer specifying the section point index for which data is required.

iElementIndices

[in|required] A sequence of integers specifying the element indices for which data is required.

iLoadCase

[in|optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.

[out] A sequence of doubles specifying the real and imaginary data for the drawer.

- o [GetHistoryResultData](#)

Returns a tuple of real and imaginary data for the drawer.

Parameters:

iDrawerNumber

[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

iSectionPoint

[in|required] An integer specifying the section point index for which data is required.

iElementIndex

[in|required] An integer specifying the element index for which data is required.

iLocalNode

[in|required] An integer specifying the local node index for which data is required.

iComponentSymbol

[in|required] A string specifying the component label for which data is required.

iLoadCase

[in|optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.

[out] A sequence of doubles specifying the real and imaginary data for the drawer.

- o [GetOrientationFieldInfo](#)

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Parameters:

[out] A FieldInfo object specifying the orientation field info.

FieldReader

This abstract base class provides the interface for each field on SIM.

Method Index

- o [__init__](#)

Abstract base class.

- o [GetFieldMaterial](#)

Returns the material name associated with the field. This is valid in case of a multi-species field or a CEL field.

- o [GetNumberOfDrawers](#)

Returns the number of drawers in the field. A drawer is a rectangular block of data per element class for element output or all the nodes for node output.

- o [GetOrientationFieldInfo](#)

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Methods

o __init__

Abstract base class.

Parameters:

None

o **GetFieldMaterial**

Returns the material name associated with the field. This is valid in case of a multi-species field or a CEL field.

Parameters:

[out] A string specifying the material name associated with the field.

o **GetNumberOfDrawers**

Returns the number of drawers in the field. A drawer is a rectangular block of data per element class for element output or all the nodes for node output.

Parameters:

[out] The number of drawers in the field.

o **GetOrientationFieldInfo**

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Parameters:

[out] A FieldInfo object specifying the orientation field info.

GenericPositionFieldReader

The GenericPositionFieldReader object is derived from the FieldReader object.

Method Index

o __init__

Creates a new instance of GenericPositionFieldReader object

o **GetDimensions**

Returns the dimensions for the drawer. A bulk drawer is a rectangular drawer of data per element class for element output and all the nodes for node output.

o **GetFieldResultData**

Returns a sequence of data for the drawer. A drawer is a rectangular block of data per element class for element output or all the nodes for node output.

o **GetHistoryResultData**

Returns a sequence of data for the drawer. A drawer is a rectangular block of data per element class for element output or all the nodes for node output.

o **GetOrientationFieldInfo**

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Methods

o __init__

Creates a new instance of GenericPositionFieldReader object

Parameters:

[out] A new instance of GenericPositionFieldReader object

o **GetDimensions**

Returns the dimensions for the drawer. A bulk drawer is a rectangular drawer of data per element class for element output and all the nodes for node output.

Parameters:

iDrawerNumber

[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of bulk drawers.

[out] An instance of the Dimension object.

o **GetFieldResultData**

Returns a sequence of data for the drawer. A drawer is a rectangular block of data per element class for element output or all the nodes for node output.

Parameters:

iDrawerNumber

[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

iDimensionDetailsList;

[in|required] A Dimensions object to query for the field drawer.

[out] A sequence of doubles specifying the data for the drawer.

o **GetHistoryResultData**

Returns a sequence of data for the drawer. A drawer is a rectangular block of data per element class for element output or all the nodes for node output.

Parameters:

iDrawerNumber

[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

iDimensionDetailsList;

[in|required] A Dimensions object to query for the field drawer.

[out] A sequence of doubles specifying the data for the drawer.

o [GetOrientationFieldInfo](#)

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Parameters:
[out] A FieldInfo object specifying the orientation field info.

SurfaceNodalFieldReader

The SurfaceNodalFieldReader object is derived from the FieldReader object.

Method Index

o [__init__](#)

Creates a new instance of SurfaceNodalField object

o [GetDimensions](#)

Returns an instance of SurfaceNodalDimension for the drawer with members: frame indices loadCases, nodeIndices, components,invariants, isDoublePrecision, isComplex generalContact, masterSurface, slaveSurface, outputSurface

o [GetFieldResultData](#)

Returns a tuple of real and imaginary data for the drawer.

o [GetHistoryResultData](#)

Returns a tuple of real and imaginary data for the drawer.

o [GetOrientationFieldInfo](#)

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Methods

o [__init__](#)

Creates a new instance of SurfaceNodalField object

Parameters:
[out] A new instance of SurfaceNodalField object

o [GetDimensions](#)

Returns an instance of SurfaceNodalDimension for the drawer with members: frame indices loadCases, nodeIndices, components,invariants, isDoublePrecision, isComplex generalContact, masterSurface, slaveSurface, outputSurface

Parameters:

iDrawerNumber
[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
[out] An instance of the SurfaceNodalDimension object with members: frame indices loadCases, nodeIndices, components,invariants, isDoublePrecision, isComplex.

o [GetFieldResultData](#)

Returns a tuple of real and imaginary data for the drawer.

Parameters:

iDrawerNumber
[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
iFrameIndex
[in|required] An integer specifying the frame index for which the data is requested; Set it to -1 if the field does not have associated frames.
iNodeIndices
[in|required] A sequence of integers specifying the node indices for which data is required.
iLoadCase
[in|optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.
[out] A sequence of doubles specifying the real and imaginary data for the drawer.

o [GetHistoryResultData](#)

Returns a tuple of real and imaginary data for the drawer.

Parameters:

iDrawerNumber
[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
iNodeIndex
[in|required] An integer specifying the node index for which data is required.
iComponentSymbol
[in|required] A string specifying the component label for which data is required.
iLoadCase
[in|optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.
[out] A sequence of doubles specifying the real and imaginary data for the drawer.

o [GetOrientationFieldInfo](#)

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Parameters:
[out] A FieldInfo object specifying the orientation field info.

TempPointElementNodalFieldReader

The TempPointElementNodalFieldReader object is derived from the FieldReader object.

Method Index

- o [__init__](#)
Creates a new instance of TempPointElementNodalField object
- o [GetDimensions](#)
Returns an instance of TempPointElementNodalDimension for the drawer with members: frame indices, loadCases, sectionPoints, elementIndices, localNodes, temperature points, components,invariants, isDoublePrecision, isComplex
- o [GetFieldResultData](#)
Returns a tuple of real and imaginary data for the drawer.
- o [GetHistoryResultData](#)
Returns a tuple of real and imaginary data for the drawer.
- o [GetOrientationFieldInfo](#)
Gets the associated orientation field info. Returns false if the field has no associated orientation.

Methods

- o [__init__](#)
Creates a new instance of TempPointElementNodalField object
Parameters:
[out] A new instance of TempPointElementNodalField object
- o [GetDimensions](#)
Returns an instance of TempPointElementNodalDimension for the drawer with members: frame indices, loadCases, sectionPoints, elementIndices, localNodes, temperature points, components,invariants, isDoublePrecision, isComplex
Parameters:
iDrawerNumber
[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
[out] An instance of the TempPointElementNodalDimension object with members: frame indices, loadCases, sectionPoints, elementIndices, localNodes, temperaturePoints, components,invariants, isDoublePrecision, isComplex.
- o [GetFieldResultData](#)
Returns a tuple of real and imaginary data for the drawer.
Parameters:
iDrawerNumber
[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
iFrameIndex
[in|required] An integer specifying the frame index for which the data is requested; Set it to -1 if the field does not have associated frames.
iSectionPoint
[in|required] An integer specifying the section point for which data is required.
iElementIndices
[in|required] A sequence of integers specifying the node indices for which data is required.
iTTemperaturePoints
[in|required] A sequence of integers specifying the temperature points for which data is required.
iLoadCase
[in|optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.
[out] A sequence of doubles specifying the real and imaginary data for the drawer.
- o [GetHistoryResultData](#)
Returns a tuple of real and imaginary data for the drawer.
Parameters:
iDrawerNumber
[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
iSectionPoint
[in|required] An integer specifying the section point index for which data is required.
iElementIndex
[in|required] An integer specifying the element index for which data is required.
iLocalNode
[in|required] An integer specifying the local node index for which data is required.
iTTemperaturePoint
[in|required] An integer specifying the temperature point for which data is required.
iComponentSymbol
[in|required] A string specifying the component label for which data is required.
iLoadCase
[in|optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.
[out] A sequence of doubles specifying the real and imaginary data for the drawer.
- o [GetOrientationFieldInfo](#)
Gets the associated orientation field info. Returns false if the field has no associated orientation.
Parameters:
[out] A FieldInfo object specifying the orientation field info.

TempPointNodalFieldReader

The TempPointNodalFieldReader object is derived from the FieldReader object.

Method Index

- o [__init__](#)
Creates a new instance of TempPointNodalField object
- o [GetDimensions](#)
Returns an instance of TempPointNodalDimension for the drawer with members: frame indices, loadCases, nodeIndices, temperature points, components,invariants, isDoublePrecision, isComplex
- o [GetFieldResultData](#)
Returns a tuple of real and imaginary data for the drawer.
- o [GetHistoryResultData](#)
Returns a tuple of real and imaginary data for the drawer.
- o [GetOrientationFieldInfo](#)
Gets the associated orientation field info. Returns false if the field has no associated orientation.

Methods

- o [__init__](#)
Creates a new instance of TempPointNodalField object
Parameters:
[out] A new instance of TempPointNodalField object
- o [GetDimensions](#)
Returns an instance of TempPointNodalDimension for the drawer with members: frame indices, loadCases, nodeIndices, temperature points, components,invariants, isDoublePrecision, isComplex
Parameters:
iDrawerNumber
 [in/required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
[out] An instance of the TempPointNodalDimension object with members: frame indices, loadCases, nodeIndices, temperaturePoints, components, invariants, isDoublePrecision, isComplex.
- o [GetFieldResultData](#)
Returns a tuple of real and imaginary data for the drawer.
Parameters:
iDrawerNumber
 [in/required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
iFrameIndex
 [in/required] An integer specifying the frame index for which the data is requested; Set it to -1 if the field does not have associated frames.
iNodeIndices
 [in/required] A sequence of integers specifying the node indices for which data is required.
iTTemperaturePoints
 [in/required] A sequence of integers specifying the temperature points for which data is required.
iLoadCase
 [in/optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.
[out] A sequence of doubles specifying the real and imaginary data for the drawer.
- o [GetHistoryResultData](#)
Returns a tuple of real and imaginary data for the drawer.
Parameters:
iDrawerNumber
 [in/required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
iNodeIndex
 [in/required] An integer specifying the node index for which data is required.
iTTemperaturePoint
 [in/required] An integer specifying the temperature point for which data is required.
iComponentSymbol
 [in/required] A string specifying the component label for which data is required.
iLoadCase
 [in/optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.
[out] A sequence of doubles specifying the real and imaginary data for the drawer.
- o [GetOrientationFieldInfo](#)
Gets the associated orientation field info. Returns false if the field has no associated orientation.
Parameters:
[out] A FieldInfo object specifying the orientation field info.

UniqueNodalFieldReader

The UniqueNodalFieldReader object is derived from the FieldReader object.

Method Index

- o [__init__](#)
Creates a new instance of UniqueNodalField object
- o [GetDimensions](#)
Returns an instance of UniqueNodalDimension for the drawer with members: frame indices, loadCases, nodeIndices, components,invariants, isDoublePrecision, isComplex

- o [GetFieldResultData](#)
Returns a tuple of real and imaginary data for the drawer.
- o [GetHistoryResultData](#)
Returns a tuple of real and imaginary data for the drawer.
- o [GetOrientationFieldInfo](#)
Gets the associated orientation field info. Returns false if the field has no associated orientation.

Methods

o __init__

Creates a new instance of UniqueNodalField object

Parameters:

[out] A new instance of UniqueNodalField object

o [GetDimensions](#)

Returns an instance of UniqueNodalDimension for the drawer with members: frame indices, loadCases, nodeIndices, components,invariants, isDoublePrecision, isComplex

Parameters:

iDrawerNumber

[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

[out] An instance of the UniqueNodalDimension object with members: frame indices, loadCases, nodeIndices, components,invariants, isDoublePrecision, isComplex.

o [GetFieldResultData](#)

Returns a tuple of real and imaginary data for the drawer.

Parameters:

iDrawerNumber

[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

iFrameIndex

[in|required] An integer specifying the frame index for which the data is requested; Set it to -1 if the field does not have associated frames.

iNodeIndices

[in|required] A sequence of integers specifying the node indices for which data is required.

iLoadCase

[in|optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is - 1.

[out] A sequence of doubles specifying the real and imaginary data for the drawer.

o [GetHistoryResultData](#)

Returns a tuple of real and imaginary data for the drawer.

Parameters:

iDrawerNumber

[in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

iNodeIndex

[in|required] An integer specifying the node index for which data is required.

iComponentSymbol

[in|required] A string specifying the component label for which data is required.

iLoadCase

[in|optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is - 1.

[out] A sequence of doubles specifying the real and imaginary data for the drawer.

o [GetOrientationFieldInfo](#)

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Parameters:

[out] A FieldInfo object specifying the orientation field info.

WholeElementFieldReader

The WholeElementFieldReader object is derived from the FieldReader object.

Method Index

- o __init__
Creates a new instance of WholeElementFieldReader object
- o [GetDimensions](#)
Returns an instance of WholeElementFieldReader for the drawer with members: frame indices, loadcase indices, element indices, Component labels, Invariants, isDoublePrecision, isComplex
- o [GetFieldResultData](#)
Returns a tuple of real and imaginary data for the drawer.
- o [GetHistoryResultData](#)
Returns a tuple of real and imaginary data for the drawer.
- o [GetOrientationFieldInfo](#)
Gets the associated orientation field info. Returns false if the field has no associated orientation.

Methods

o [__init__](#)

Creates a new instance of WholeElementFieldReader object

Parameters:

[out] A new instance of WholeElementFieldReader object

o [GetDimensions](#)

Returns an instance of WholeElementFieldReader for the drawer with members: frame indices, loadcase indices, element indices, Component labels, Invariants, isDoublePrecision, isComplex

Parameters:

iDrawerNumber

[in/required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

[out] An instance of the WholeElementFieldReader object with members: frame indices, loadcase indices, element indices, Component labels, Invariants, isDoublePrecision, isComplex.

o [GetFieldResultData](#)

Returns a tuple of real and imaginary data for the drawer.

Parameters:

iDrawerNumber

[in/required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

iFrameIndex

[in/required] An integer specifying the frame index for which the data is requested; Set it to -1 if the field does not have associated frames.

iElementIndices

[in/required] A sequence of integers specifying the element indices for which data is required.

iLoadCase

[in/optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.

[out] A sequence of doubles specifying the real and imaginary data for the drawer.

o [GetHistoryResultData](#)

Returns a tuple of real and imaginary data for the drawer.

Parameters:

iDrawerNumber

[in/required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

iElementIndex

[in/required] An integer specifying the element index for which data is required.

iComponentSymbol

[in/required] A string specifying the component label for which data is required.

iLoadCase

[in/optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.

[out] A sequence of doubles specifying the real and imaginary data for the drawer.

o [GetOrientationFieldInfo](#)

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Parameters:

[out] A FieldInfo object specifying the orientation field info.

WholeModelFieldReader

The WholeModelFieldReader object is derived from the FieldReader object.

Method Index

o [__init__](#)

Creates a new instance of WholeModelField object

o [GetDimensions](#)

Returns an instance of WholeModelDimension for the drawer with members: frame indices, loadCases, components,invariants, isDoublePrecision, isComplex

o [GetFieldResultData](#)

Returns a tuple of real and imaginary data for the drawer.

o [GetHistoryResultData](#)

Returns a tuple of real and imaginary data for the drawer.

o [GetOrientationFieldInfo](#)

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Methods

o [__init__](#)

Creates a new instance of WholeModelField object

Parameters:

[out] A new instance of WholeModelField object

o [GetDimensions](#)

Returns an instance of WholeModelDimension for the drawer with members: frame indices, loadCases, components,invariants, isDoublePrecision, isComplex

Parameters:

iDrawerNumber

[in/required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.

[out] An instance of the WholeModelDimension object with members: frame indices, loadCases, components,invariants, isDoublePrecision, isComplex.

o [GetFieldResultData](#)

Returns a tuple of real and imaginary data for the drawer.

Parameters:

iDrawerNumber	[in required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
iFrameIndex	[in required] An integer specifying the frame index for which the data is requested; Set it to -1 if the field does not have associated frames.
iLoadCase	[in/optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is - 1.
[out]	A sequence of doubles specifying the real and imaginary data for the drawer.

o [GetHistoryResultData](#)

Returns a tuple of real and imaginary data for the drawer.

Parameters:

iDrawerNumber	[in required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
iComponentSymbol	[in required] A string specifying the component label for which data is required.
iLoadCase	[in/optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is - 1.
[out]	A sequence of doubles specifying the real and imaginary data for the drawer.

o [GetOrientationFieldInfo](#)

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Parameters:

[out]	A FieldInfo object specifying the orientation field info.
-------	---

WholeRegionFieldReader

The WholeRegionFieldReader object is derived from the FieldReader object.

Method Index

o [__init__](#)

Creates a new instance of WholeRegionField object

o [GetDimensions](#)

Returns an instance of WholeRegionDimension for the drawer with members: frame indices, loadCases, regionIndices, components,invariants, isDoublePrecision, isComplex

o [GetFieldResultData](#)

Returns a tuple of real and imaginary data for the drawer.

o [GetHistoryResultData](#)

Returns a tuple of real and imaginary data for the drawer.

o [GetOrientationFieldInfo](#)

Gets the associated orientation field info. Returns false if the field has no associated orientation.

o [GetRegionData](#)

Returns the node or element set indices along with the set name for the given region index.

Methods

o [__init__](#)

Creates a new instance of WholeRegionField object

Parameters:

[out]	A new instance of WholeRegionField object
-------	---

o [GetDimensions](#)

Returns an instance of WholeRegionDimension for the drawer with members: frame indices, loadCases, regionIndices, components,invariants, isDoublePrecision, isComplex

Parameters:

iDrawerNumber	[in required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
[out]	An instance of the WholeRegionDimension object with members: frame indices, loadCases, regionIndices, components,invariants, isDoublePrecision, isComplex.

o [GetFieldResultData](#)

Returns a tuple of real and imaginary data for the drawer.

Parameters:

iDrawerNumber	[in required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.
iFrameIndex	[in required] An integer specifying the frame index for which the data is requested; Set it to -1 if the field does not have associated frames.
iRegionIndices	[in required] A sequence of integers specifying the node or element indices for which data is required.
iLoadCase	[in/optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is - 1.
[out]	A sequence of doubles specifying the real and imaginary data for the drawer.

o GetHistoryResultData

Returns a tuple of real and imaginary data for the drawer.

Parameters:

```
iDrawerNumber  
    [in|required] An integer specifying the drawer number; Must be non-negative and less than the number of drawers.  
iRegionIndex  
    [in|required] An integer specifying the node index for which data is required.  
iComponentSymbol  
    [in|required] A string specifying the component label for which data is required.  
iLoadCase  
    [in/optional] An integer specifying the load case index, must be specified when load case exists in dimension. The default value is -1.  
[out] A sequence of doubles specifying the real and imaginary data for the drawer.
```

o GetOrientationFieldInfo

Gets the associated orientation field info. Returns false if the field has no associated orientation.

Parameters:

```
[out] A FieldInfo object specifying the orientation field info.
```

o GetRegionData

Returns the node or element set indices along with the set name for the given region index.

Parameters:

```
iRegionIndex  
    [in|required] Region index.  
[out] An instance of RegionData class with members: regionIndex, nsetName, nodeIndices, elsetName, elementIndices.
```

FieldInfo

This struct is intended to capture the basic field info that will help uniquely identify the field on SIM.

Method Index

o __init__

Create a new FieldInfo class.

Methods

o __init__

Parameters:
None

SimResultsReaderException

The SimResultsReaderException object is derived from the Exception object.

Method Index

o __init__

Creates a new instance of SimResultsReaderException

Methods

o __init__

Parameters:
None

SIMResultsWriter

This class is intended to make it as easy as possible to write results to SIM that will be compatible with HPViz.

Method Index

o __init__

Creates a new instance of SIMResultsWriter object

o AddDisplayBodyConstraint

Creates a display body constraint for the specified set of elements.

- o [AddElements](#)
Adds the element data for mesh to the document.
- o [AddFieldType](#)
Adds a new user defined fieldType to document.
- o [AddFrame](#)
Adds a frame to the step.
- o [AddLoadCase](#)
Creates a loadcase for the step.
- o [AddModifiedFieldType](#)
Copies a existing fieldType to a new user defined fieldType, adds to the document.
- o [AddStep](#)
Creates a step.
- o [AddUserElementLabels](#)
Returns a list of flat element labels for the given vector of user element labels.
- o [AddUserNodeLabels](#)
Returns a list of flat node labels for the given vector of user node labels..
- o [CreateBeamSection](#)
Creates beam section, element class and assigns material to the section. It is expected that an element set here defines an element class; returns an error if element set contains elements of multiple element specs.
- o [CreateCompositeShellSection](#)
Creates composite shell section, element class and assigns material to the section. It is expected that an element set here defines an element class; returns an error if element set contains elements of multiple element specs.
- o [CreateContinuumSection](#)
Creates continuum section, element class and assigns material to the section. It is expected that an element set here defines an element class; returns an error if element set contains elements of multiple element specs.
- o [CreateDocument](#)
Creates a SIM document. If there is any existing document with same name, it will be overwritten.
- o [CreateHomogeneousShellSection](#)
Creates homogeneous shell section, element class and assigns material to the section. It is expected that an element set here defines an element class; returns an error if element set contains elements of multiple element specs.
- o [CreateMaterial](#)
Creates material.
- o [CreatePermutedElementSet](#)
Creates permuted set of elements.
- o [CreatePermutedNodeSet](#)
Creates permuted set of nodes.
- o [Finalize](#)
Saves and closes the SIM document; must be called only once.
- o [GetDocument](#)
Returns the underlying SIMDocument object, for advanced usage with SIM API.
- o [GetFieldTypeFromOutputVariable](#)
Gets a fieldType name for output variable.
- o [InitFieldResult](#)
Initializes a field result.
- o [InitHistoryResult](#)
Initializes a history result.
- o [InitOrientation](#)
Creates Orientation.
- o [Initialize](#)
Initializes ResultsWriter, this method must be called before any other method.
- o [LoadDocument](#)
Loads an existing SIM document. ResultsWriter expects certain state of the document, so this options should be used with caution.
- o [SetGeometricNonlinearityOption](#)
Sets the geometric nonlinearity option for the step.
- o [SetUnitSystem](#)
Set the unit system.
- o [SetUserElementLabelsMap](#)
Set the user element label to flat label mapping if it is known ahead of time.
- o [SetUserNodeLabelsMap](#)
Set the user node label to flat label mapping if it is known ahead of time.
- o [WriteNodes](#)
Writes the node coordinate data for mesh (or part of mesh) to the document. Should give better performance if data for all nodes is written at once.

Methods

o __init__

Creates a new instance of SIMResultsWriter object

Parameters:

[out] A new instance of SIMResultsWriter object

o AddDisplayBodyConstraint

Creates a display body constraint for the specified set of elements.

Parameters:

```
iConstraintName  
    [in|required] A string specifying the constraint name for the display body constraint; must be a unique.  
iConstraintDescription  
    [in|required] A string specifying the description.  
iElsetName  
    [in|required] A string specifying the element set name to be constrained as a display body.  
iRefNodeSetName  
    [in|required] A string specifying the reference node set to which the display body is connected to. There can be maximum of 3 nodes in the node set and the set must be permuted.
```

o AddElements

Adds the element data for mesh to the document.

Parameters:

```
iElementSpec  
    [in|required] A string specifying the elementSpec. Refer to "SIM predefined string" section of "Writing SIM Results Using Python" user documentation.  
iElementIndices  
    [in|required] A sequence of integers specifying the element indices.  
iConnectivity  
    [in|required] A sequence of integers specifying the connectivity for each element specified in iElementIndices; must be of length iNumElements
```

o AddFieldType

Adds a new user defined fieldType to document.

Parameters:

```
iFieldType  
    [in|required] A string specifying the FieldType to be added to SIM document.  
iFieldTypeDescription  
    [in|required] A string specifying the description.  
iAlgebraicType  
    [in|required] A string specifying the AlgebraicType.  
iPhysicalDimension  
    [in|required] A string specifying the PhysicalDimension e.g. MLT02; refer to "SIM predefined string" section of "Writing SIM Results Using Python" user documentation.  
iDsMagnitude  
    [in|required] A string specifying the dsMagnitude string to help HPViz process the field types e.g FORCE.  
iNonNegative  
    [in|optional] A boolean specifying if the field type can be negative or not. This can only be true for a SCALAR or TENSOR2_GENERAL.
```

o AddFrame

Adds a frame to the step.

Parameters:

```
iStepName  
    [in|required] A string specifying the step name for the frame; must be already created.  
iFrameIndex  
    [in|required] An integer specifying the frame Index to be added; must not be already defined.  
iFrameValues  
    [in|required] A double specifying the frame value.
```

o AddLoadCase

Creates a loadcase for the step.

Parameters:

```
iStepName  
    [in|required] A string specifying the step name for the load case; must be already created.  
iCaseName  
    [in|required] A string specifying the loadcase name to be created; must be unique across all steps.
```

o AddModifiedFieldType

Copies a existing fieldType to a new user defined fieldType, adds to the document.

Parameters:

```
iOriginalFieldType  
    [in|required] A string specifying the original field type to be copied, must exist on SIM document.  
iFieldType  
    [in|required] A string specifying the FieldType to be added to SIM document.  
iFieldTypeDescription  
    [in|required] A string specifying the description.
```

o AddStep

Creates a step.

Parameters:

```
iStepName  
    [in|required] A string specifying the step name to be created; must be unique.  
iSteptypeName  
    [in|required] A string specifying the type name of the step to be created; all step type names are available in "SIM predefined string" section of "Writing SIM Results Using Python" user documentation.  
iStepDescription
```

iFrameValues
[in/required] A sequence of doubles specifying the frame values.

o AddUserElementLabels

Returns a list of flat element labels for the given vector of user element labels.

Parameters:

iInstanceName
[in/required] A string specifying the instance name.
iUserLabels
[in/required] A sequence of integers specifying the user element labels.
[out] A sequence of integers specifying the flat element labels.

o AddUserNodeLabels

Returns a list of flat node labels for the given vector of user node labels..

Parameters:

iInstanceName
[in/required] A string specifying the instance name.
iUserLabels
[in/required] A sequence of integers specifying the user node labels.
[out] A sequence of integers specifying the flat node labels.

o CreateBeamSection

Creates beam section, element class and assigns material to the section. It is expected that an element set here defines an element class; returns an error if element set contains elements of multiple element specs.

Parameters:

iElsetName
[in/required] A string specifying the element set name; must be already created.
iMaterialName
[in/required] A string specifying the material name to be assigned to section; must be already created.
iProfileType
[in/required] A string specifying the beam Profile type.
iNumDirSectionPoints
[in/required] A sequence of integers specifying the number of section points in each direction.

o CreateCompositeShellSection

Creates composite shell section, element class and assigns material to the section. It is expected that an element set here defines an element class; returns an error if element set contains elements of multiple element specs.

Parameters:

iElsetName
[in/required] A string specifying the element set name; must be already created.
iNumLayers
[in/required] An integer specifying the number of layers.
iMaterialNames
[in/required] A sequence of strings specifying the name of Materials to be assigned for each layer in section; must be already created.
iPlyNames
[in/required] A sequence of strings specifying the ply name for each layer.
iNumSectionPoints
[in/required] An integer specifying the number of section points, same for all layers.
iLayerThickness
[in/required] A sequence of doubles specifying the thickness for each layer.

o CreateContinuumSection

Creates continuum section, element class and assigns material to the section. It is expected that an element set here defines an element class; returns an error if element set contains elements of multiple element specs.

Parameters:

iElsetName
[in/required] A string specifying the element set name; must be already created.
iMaterialName
[in/required] A string specifying the material name to be assigned to section; must be already created.

o CreateDocument

Creates a SIM document. If there is any existing document with same name, it will be overwritten.

Parameters:

iDocName
[in/required] A string specifying the document name.

o CreateHomogeneousShellSection

Creates homogeneous shell section, element class and assigns material to the section. It is expected that an element set here defines an element class; returns an error if element set contains elements of multiple element specs.

Parameters:

iElsetName
[in/required] A string specifying the element set name; must be already created.
iMaterialName
[in/required] A string specifying the material name to be assigned to section; must be already created.
iNumSectionPoints
[in/required] An integer specifying the number of section points.
iThickness
[in/optional] A double specifying the section thickness. The default value is 1.0.

o CreateMaterial

Creates material.

Parameters:

iMaterialName
[in/required] A string specifying the material name; must be unique.

o CreatePermutedElementSet

Creates permuted set of elements.

Parameters:

iSetName
[in|required] A string specifying the set name to be created; must be unique for both element and node sets.
iMembers
[in|required] A sequence of integers specifying the members; order is maintained, no duplicates.

o CreatePermutedNodeSet

Creates permuted set of nodes.

Parameters:

iSetName
[in|required] A string specifying the set name to be created; must be unique for both node and element sets.
iMembers
[in|required] A sequence of integers specifying the members; order is maintained, no duplicates.

o Finalize

Saves and closes the SIM document; must be called only once.

Parameters:

None

o GetDocument

Returns the underlying SIMDocument object, for advanced usage with SIM API.

Parameters:

[out] A SIMDocument object specifying the SIMDocument object.

o GetFieldTypeFromOutputVariable

Gets a fieldType name for output variable.

Parameters:

iOutputVariable
[in|required] A string specifying output variable name for which fieldType is needed.
[out] A string specifying the FieldType.

o InitFieldResult

Initializes a field result.

Parameters:

iStepName
[in|required] A string specifying the step name for the result; must be already created.
iFieldType
[in|required] A string specifying the fieldType name. Refer to "SIM predefined string" section of "Writing SIM Results Using Python" user documentation.
ioField
[in|required] A Field object specifying the field object which should be used to add data.

o InitHistoryResult

Initializes a history result.

Parameters:

iStepName
[in|required] A string specifying the step name for the result; must be already created.
iFieldType
[in|required] A string specifying the fieldType name; must be from "SIM predefined string" section of "Writing SIM Results Using Python" user documentation.
ioField
[in|required] A Field object specifying the field object which should be used to add data.

o InitOrientation

Creates Orientation.

Parameters:

iOrientationName
[in|required] Orientation name to be created; must be unique.
ioOrientation
[in|required] Orientation object which should be used to add data.

o Initialize

Initializes ResultsWriter, this method must be called before any other method.

Parameters:

iMaxChunkSizeInMB
[in|optional] An integer specifying the max chunk size in MB. A chunk is the unit of I/O for bulk data. The default value is 1.

o LoadDocument

Loads an existing SIM document. ResultsWriter expects certain state of the document, so this options should be used with caution.

Parameters:

iDocName
[in|required] A string specifying the document name.

o SetGeometricNonlinearityOption

Sets the geometric nonlinearity option for the step.

Parameters:

iStepName
[in|required] A string specifying the step name for the option; must be already created.

o SetUnitSystem

Set the unit system.

Parameters:

```
iUnitSystemName
    [in/required] A string specifying the name of the unit system.
iLengthToSIConvFactor
    [in/optional] A double specifying the length to SI conversion factor (default = 1.0).
iMassToSIConvFactor
    [in/optional] A double specifying the mass to SI conversion factor (default = 1.0).
iTTimeToSIConvFactor
    [in/optional] A double specifying the time to SI conversion factor (default = 1.0).
iElectricCurrentToSIConvFactor
    [in/optional] A double specifying the electric current to SI conversion factor (default = 1.0).
iTTemperatureToSIConvFactor
    [in/optional] A double specifying the temperature to SI conversion factor (default = 1.0).
iTTemperatureToSIOffset
    [in/optional] A double specifying the temperature to SI offset (default = 0.0).
iAmtOfSubToSIConvFactor
    [in/optional] A double specifying the amount of substance to SI conversion factor (default = 1.0).
iLumIntensityToSIConvFactor
    [in/optional] A double specifying the Luminous intensity to SI conversion factor (default = 1.0).
iAngleToSIConvFactor
    [in/optional] A double specifying the angle to SI conversion factor (default = 1.0).
iSolidAngleToSIConvFactor
    [in/optional] A double specifying the solid angle to SI conversion factor (default = 1.0).
```

o **SetUserElementLabelsMap**

Set the user element label to flat label mapping if it is known ahead of time.

Parameters:

```
iInstanceName
    [in/required] A string specifying the instance name.
iUserLabels
    [in/required] A sequence of integers specifying the user element labels.
iFlatLabels
    [in/required] A sequence of integers specifying the flat element labels.
```

o **SetUserNodeLabelsMap**

Set the user node label to flat label mapping if it is known ahead of time.

Parameters:

```
iInstanceName
    [in/required] A string specifying the instance name.
iUserLabels
    [in/required] A sequence of integers specifying the user node labels.
iFlatLabels
    [in/required] A sequence of integers specifying the flat node labels.
```

o **WriteNodes**

Writes the node coordinate data for mesh (or part of mesh) to the document. Should give better performance if data for all nodes is written at once.

Parameters:

```
iNumSpaceDims
    [in/required] An integer specifying the number of space dimension for model: 2 for 2D and 3 for 3D.
iNodeIndices
    [in/required] A sequence of integers specifying the node indices.
iCoords
    [in/required] A sequence of doubles specifying the coordinates; must be of length iNumNodes*iNumSpaceDims.
```

ElemCentroidField

The ElemCentroidField object is derived from the FieldWriter object.

Method Index

o [init](#)

Creates a new instance of ElemCentroidField object

o [AddFrameData](#)

Adds float data for a frame for a results field, dimensions must be defined before this method call.

o [DefineDimensions](#)

Defines dimensions for ElemCentroidField.

Methods

o [__init__](#)

Creates a new instance of ElemCentroidField object

Parameters:
[out] A new instance of ElemCentroidField object

o [AddFrameData](#)

Adds float data for a frame for a results field, dimensions must be defined before this method call.

Parameters:
iFrameIndex
 [in/required] An integer specifying the frame index.
iSectionPoint

```

[in/required] An integer specifying the section point.
iElsetName
[in/required] A string specifying the element set name; must be same as used in DefineDimensions or subset of it.
iData
[in/optional] A sequence of floats specifying the real data; must be in the order #Elements *#Components. The default is an empty sequence.
iImaginaryData
[in/optional] A sequence of floats specifying the imaginary data, must be in the order #Elements *#Components. The default value is an empty
sequence.
iLoadCaseName
[in/optional] A string specifying the load case name; must be used if a step has a load case. The default value is "".

```

o DefineDimensions

Defines dimensions for ElemCentroidField.

Parameters:

```

iElsetName
[in/required] A string specifying the element set for element dimension; all elements must be from same element class.
iSectionPoints
[in/required] A sequence of integers specifying the section points for section point dimension.
iComponentSymbols
[in/required] A sequence of strings specifying the components for component dimension.
iInvariantSymbols
[in/required] A sequence of strings specifying the invariant symbols; must be from "SIM predefined string" section of "Writing SIM Results Using
Python" user documentation.
iComplex
[in/required] A boolean specifying the complex - true if field has real and imaginary data.
iDoublePrecision
[in/optional] A boolean specifying the double precision data. The default value is false.

```

ElemFaceField

The ElemFaceField object is derived from the FieldWriter object.

Method Index

o [__init__](#)

Creates a new instance of ElemFaceField object

o [AddFrameData](#)

Adds float data for a frame for a results field; dimensions must be defined before this method call.

o [AddHistoryData](#)

Adds float data for frames for a history field, dimensions must be defined before this method call.

o [DefineDimensions](#)

Defines dimensions for ElemFaceField.

Methods

o [__init__](#)

Creates a new instance of ElemFaceField object

Parameters:
[out] A new instance of ElemFaceField object

o [AddFrameData](#)

Adds float data for a frame for a results field; dimensions must be defined before this method call.

Parameters:

```

iFrameIndex
[in/required] An integer specifying the frame index.
iElementSetName
[in/required] A string specifying the element set name; must be same as used in DefineDimensions or subset of it.
iData
[in/optional] A sequence of floats specifying the real data; must be in the order #Elements *#Components. The default is an empty sequence.
iImaginaryData
[in/optional] A sequence of floats specifying the imaginary data, must be in the order #Elements *#Components. The default value is an empty
sequence.
iLoadCaseName
[in/optional] A string specifying the load case name; must be used if a step has a load case. The default value is "".

```

o [AddHistoryData](#)

Adds float data for frames for a history field, dimensions must be defined before this method call.

Parameters:

```

iFrameIndices
[in/required] A sequence of integers specifying the frame indices. This sequence can be a subset of the 'iFrameValues' in AddStep method.
iElementSetName
[in/required] A string specifying the element set name; must be same as used in DefineDimensions or subset of it.
iData
[in/optional] A sequence of floats specifying the real data; must be in the order #Elements *#Components. The default is an empty sequence.
iImaginaryData
[in/optional] A sequence of floats specifying the imaginary data, must be in the order #Elements *#Components. The default value is an empty
sequence.
iLoadCaseName
[in/optional] A string specifying the load case name; must be used if a step has a load case. The default value is "".

```

o DefineDimensions

Defines dimensions for ElemFaceField.

Parameters:

```
iElementSetName
    [in/required] A string specifying the element set for element dimension.
iFaceID
    [in/required] Face index.
iComponentSymbols
    [in/required] A sequence of strings specifying the components for component dimension.
iInvariantSymbols
    [in/required] A sequence of strings specifying the invariant symbols; must be from "SIM predefined string" section of "Writing SIM Results Using Python" user documentation.
iComplex
    [in/required] A boolean specifying the complex - true if field has real and imaginary data.
iDoublePrecision
    [in/optional] A boolean specifying the double precision data. The default value is false.
```

ElemIntegrationPointField

The ElemIntegrationPointField object is derived from the FieldWriter object.

Method Index

o [init](#)

Creates a new instance of ElemIntegrationPointField object

o [AddFrameData](#)

Adds float data for a frame for a results field, dimensions must be defined before this method call.

o [DefineDimensions](#)

Defines dimensions for ElemIntegrationPointField.

Methods

o [__init__](#)

Creates a new instance of ElemIntegrationPointField object

Parameters:

```
[out] A new instance of ElemIntegrationPointField object
```

o [AddFrameData](#)

Adds float data for a frame for a results field, dimensions must be defined before this method call.

Parameters:

```
iFrameIndex
    [in/required] An integer specifying the frame index.
iSectionPoint
    [in/required] An integer specifying the section point.
iElsetName
    [in/required] A string specifying the element set name; must be same as used in DefineDimensions or subset of it.
iData
    [in/optional] A sequence of floats specifying the real data; must be in the order #Elements *#IntegrationPts *#Components. The default is an empty sequence.
iImaginaryData
    [in/optional] A sequence of floats specifying the imaginary data, must be in the order #Elements *#IntegrationPts *#Components. The default value is an empty sequence.
iLoadCaseName
    [in/optional] A string specifying the load case name; must be used if a step has a load case. The default value is "".
```

o [DefineDimensions](#)

Defines dimensions for ElemIntegrationPointField.

Parameters:

```
iElsetName
    [in/required] A string specifying the element set for element dimension; all elements must be from same element class.
iSectionPoints
    [in/required] A sequence of integers specifying the section points for section point dimension.
iNumIntegrationPts
    [in/required] An integer specifying the number of integration points; must be greater than 0.
iComponentSymbols
    [in/required] A sequence of strings specifying the components for component dimension.
iInvariantSymbols
    [in/required] A sequence of strings specifying the invariant symbols; must be from "SIM predefined string" section of "Writing SIM Results Using Python" user documentation.
iComplex
    [in/required] A boolean specifying the complex - true if field has real and imaginary data.
iDoublePrecision
    [in/optional] A boolean specifying the double precision data. The default value is false.
```

ElemNodalField

The ElemNodalField object is derived from the FieldWriter object.

Method Index

- o [__init__](#)
Creates a new instance of ElemNodalField object
- o [AddFrameData](#)
Adds float data for a frame for a results field, dimensions must be defined before this method call.
- o [DefineDimensions](#)
Defines dimensions for an element class of ElemNodalField.

Methods

- o [__init__](#)
Creates a new instance of ElemNodalField object
Parameters:
[out] A new instance of ElemNodalField object
 - o [AddFrameData](#)
Adds float data for a frame for a results field, dimensions must be defined before this method call.
Parameters:
iFrameIndex [in|required] An integer specifying the frame index.
iSectionPoint [in|required] An integer specifying the section point. Set it to -1 if there are no section points.
iElsetName [in|required] A string specifying the element set name; must be same as used in DefineDimensions or subset of it.
iData [in/optional] A sequence of floats specifying the real data; must be in the order #Elements *#Nodes *#Components. The default is an empty sequence.
iImaginaryData [in/optional] A sequence of floats specifying the imaginary data, must be in the order #Elements *#Nodes *#Components. The default value is an empty sequence.
iLoadCaseName [in/optional] A string specifying the load case name; must be used if a step has a load case. The default value is "".
 - o [DefineDimensions](#)
Defines dimensions for an element class of ElemNodalField.
Parameters:
iElsetName [in|required] A string specifying the element set for element dimension; all elements must be from same element class.
iNodesPerElement [in|required] An integer specifying the number of nodes for nodal dimension; must be greater than 0.
iSectionPoints [in|required] A sequence of integers specifying the section points for section point dimension. Set it to empty sequence when there are no section points.
iComponentSymbols [in|required] A sequence of strings specifying the components for component dimension.
iInvariantSymbols [in|required] A sequence of strings specifying the invariant symbols; must be from "SIM predefined string" section of "Writing SIM Results Using Python" user documentation.
iComplex [in|required] A boolean specifying the complex - true if field has real and imaginary data.
iDoublePrecision [in/optional] A boolean specifying the double precision data. The default value is false.
-

UniqueNodalField

The UniqueNodalField object is derived from the FieldWriter object.

Method Index

- o [__init__](#)
Creates a new instance of UniqueNodalField object
- o [AddFrameData](#)
Adds float data for a frame for a results field; dimensions must be defined before this method call.
- o [AddHistoryData](#)
Adds float data for frames for a history field, dimensions must be defined before this method call.
- o [DefineDimensions](#)
Defines dimensions for UniqueNodalField.

Methods

- o [__init__](#)
Creates a new instance of UniqueNodalField object
Parameters:
[out] A new instance of UniqueNodalField object

```
[out] A new instance of UniqueNodalField object
```

o AddFrameData

Adds float data for a frame for a results field; dimensions must be defined before this method call.

Parameters:

```
iFrameIndex
    [in|required] An integer specifying the frame index.
iNodeSetName
    [in|required] A string specifying the node set name; must be same as used in DefineDimensions or subset of it.
iData
    [in/optional] A sequence of floats specifying the real data; must be in the order #Nodes *#Components. The default is an empty sequence.
iImaginaryData
    [in/optional] A sequence of floats specifying the imaginary data, must be in the order #Nodes *#Components. The default value is an empty sequence.
iLoadCaseName
    [in/optional] A string specifying the load case name; must be used if a step has a load case. The default value is "".
```

o AddHistoryData

Adds float data for frames for a history field, dimensions must be defined before this method call.

Parameters:

```
iFrameIndices
    [in|required] A sequence of integers specifying the frame indices. This sequence can be a subset of the 'iFrameValues' in AddStep method.
iNodeSetName
    [in|required] A string specifying the node set name; must be same as used in DefineDimensions or subset of it.
iData
    [in/optional] A sequence of floats specifying the real data; must be in the order #Nodes *#Components. The default is an empty sequence.
iImaginaryData
    [in/optional] A sequence of floats specifying the imaginary data, must be in the order #Nodes *#Components. The default value is an empty sequence.
iLoadCaseName
    [in/optional] A string specifying the load case name; must be used if a step has a load case. The default value is "".
```

o DefineDimensions

Defines dimensions for UniqueNodalField.

Parameters:

```
iNodeSetName
    [in|required] A string specifying the node set for node dimension.
iComponentSymbols
    [in|required] A sequence of strings specifying the components for component dimension.
iInvariantSymbols
    [in|required] A sequence of strings specifying the invariant symbols; must be from "SIM predefined string" section of "Writing SIM Results Using Python" user documentation.
iComplex
    [in|required] A boolean specifying the complex - true if field has real and imaginary data.
iDoublePrecision
    [in/optional] A boolean specifying the double precision data. The default value is false.
```

WholeElemField

The WholeElemField object is derived from the FieldWriter object.

Method Index

o __init__

Creates a new instance of WholeElemField object

o AddFrameData

Adds float data for a frame for a results field, dimensions must be defined before this method call.

o DefineDimensions

Defines dimensions for WholeElemField.

Methods

o __init__

Creates a new instance of WholeElemField object

Parameters:

```
[out] A new instance of WholeElemField object
```

o AddFrameData

Adds float data for a frame for a results field, dimensions must be defined before this method call.

Parameters:

```
iFrameIndex
    [in|required] An integer specifying the frame index.
iElsetName
    [in|required] A string specifying the element set name; must be same as used in DefineDimensions or subset of it.
iData
    [in/optional] A sequence of floats specifying the real data; must be in the order #Elements *#Components. The default is an empty sequence.
iImaginaryData
    [in/optional] A sequence of floats specifying the imaginary data, must be in the order #Elements *#Components. The default value is an empty sequence.
iLoadCaseName
```

[in/optional] A string specifying the load case name; must be used if a step has a load case. The default value is "".

o DefineDimensions

Defines dimensions for WholeElemField.

Parameters:

```
iElsetName      [in/required] A string specifying the element set for element dimension; all elements must be from same element class.
iComponentSymbols [in/required] A sequence of strings specifying the components for component dimension.
iInvariantSymbols [in/required] A sequence of strings specifying the invariant symbols; must be from "SIM predefined string" section of "Writing SIM Results Using Python" user documentation.
iComplex        [in/required] A boolean specifying the complex - true if field has real and imaginary data.
iDoublePrecision [in/optional] A boolean specifying the double precision data. The default value is false.
```

CartesianOrientation

The CartesianOrientation object is derived from the FieldWriter object.

Method Index

o __init__

Creates a new instance of CartesianOrientation object

o Define

Define Cartesian Orientation.

o Validate

Check if the Orientation is defined and valid

Methods

o __init__

Creates a new instance of CartesianOrientation object

Parameters:
[out] A new instance of CartesianOrientation object

o Define

Define Cartesian Orientation.

Parameters:

```
iOrg           [in/required] Origin coordinates [x,y,z]
iQuaternion0   [in/required] Quaternion component r
iQuaternion1   [in/required] Quaternion component u
iQuaternion2   [in/required] Quaternion component v
iQuaternion3   [in/required] Quaternion component w
```

o Validate

Check if the Orientation is defined and valid

Parameters:
None

CylindricalOrientation

The CylindricalOrientation object is derived from the FieldWriter object.

Method Index

o __init__

Creates a new instance of CylindricalOrientation object

o Define

Define Cylindrical Orientation.

o Validate

Check if the Orientation is defined and valid

Methods

o __init__

Creates a new instance of CylindricalOrientation object

Parameters:

[out] A new instance of CylindricalOrientation object

o Define

Define Cylindrical Orientation.

Parameters:

iPointA
[in|required] Coordinate of First Axis Point [x,y,z]
iPointB
[in|required] Coordinate of Second Axis Point [x,y,z]

o Validate

Check if the Orientation is defined and valid

Parameters:

None

FieldWriter

The abstract base of all field classes. Fields in SIM are represented as multi-dimensional arrays.

Method Index

o [SetOrientationField](#)

Sets orientation field for this field object.

Methods

o SetOrientationField

Sets orientation field for this field object.

Parameters:

iOrientationField
[in|required] Orientation field to be associated with this field.

SIMDocument

Method Index

o __init__

Creates a new instance of SIMDocument object

Methods

o __init__

Creates a new instance of SIMDocument object

Parameters:

[out] A new instance of SIMDocument object

SimResultsWriterException

The SimResultsWriterException object is derived from the Exception object.

Method Index

o __init__

Creates a new instance of SimResultsWriterException object

Methods

o __init__

Creates a new instance of SimResultsWriterException object

Parameters:

[out] A new instance of SimResultsWriterException object

SphericalOrientation

The SphericalOrientation object is derived from the FieldWriter object.

Method Index

- o [__init__](#)
Creates a new instance of SphericalOrientation object

- o [Define](#)
Define Spherical Orientation.

- o [Validate](#)
Check if the Orientation is defined and valid

Methods

o [__init__](#)

Creates a new instance of SphericalOrientation object

Parameters:
[out] A new instance of SphericalOrientation object

o [Define](#)

Define Spherical Orientation.

Parameters:
iPointA
[in|required] Coordinate of First Axis Point [x,y,z]
iPointB
[in|required] Coordinate of Second Axis Point [x,y,z]

o [Validate](#)

Check if the Orientation is defined and valid

Parameters:
None

SIMDocUtils

This class includes all the utility methods needed for a SIM document.

Method Index

- o [ConvertDocument](#)
Upgrades or downgrades the given SIM document.
- o [DowngradeRequired](#)
Query to determine if the document specified by the supplied path requires downgrade.
- o [UpgradeRequired](#)
Query to determine if the document specified by the supplied path requires upgrade.

Methods

o [ConvertDocument](#)

Upgrades or downgrades the given SIM document.

Parameters:
iDocName
[in|required] Name of the document to upgrade/downgrade.
iUpgradeDocName
[in|required] Name of the upgraded/downgraded document.

o [DowngradeRequired](#)

Query to determine if the document specified by the supplied path requires downgrade.

Parameters:
iDocName
[in|required] Name of the document to query. True if the document requires downgrade.
[out] True if the document requires downgrade.

o [UpgradeRequired](#)

Query to determine if the document specified by the supplied path requires upgrade.

Parameters:
iDocName
[in|required] Name of the document to query. True if the document requires upgrade.
[out] True if the document requires upgrade.

SimDocUtilsException

The SimDocUtilsException object is derived from the Exception object.

Method Index

- o [__init__](#)
Creates a new instance of SimDocUtilsException

Methods

- o [__init__](#)

Parameters:
None