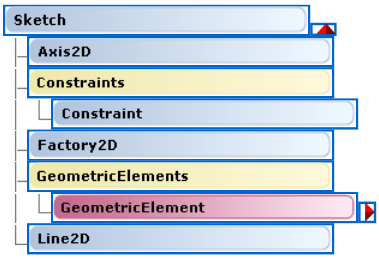
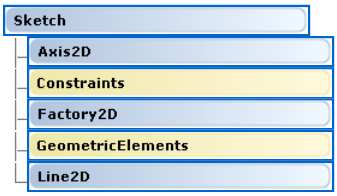
# Sketcher Object Model Map

See Also Legend




# Sketch Object

See Also Legend UseCases Properties Methods

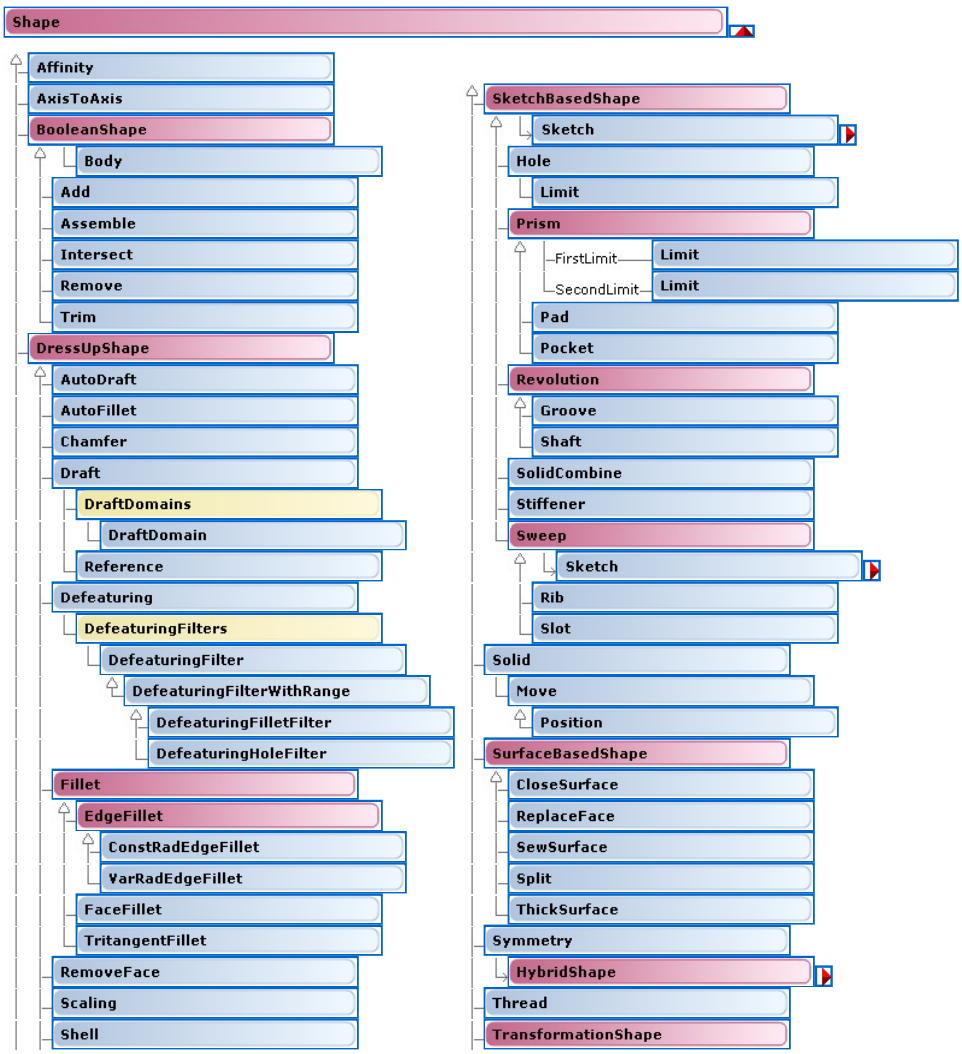


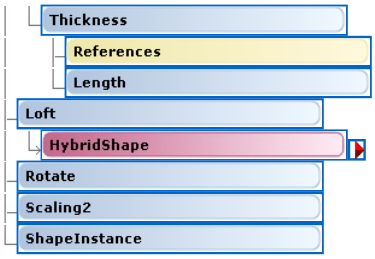

The **Sketch** object contains 2D geometric elements that define the sketch. These elements are created using the **Factory2D** object and contained in a **GeometricElements** collection aggregated by the sketch. To create 2D geometric elements, you first need to "open the sketch edition" using the **OpenEdition** method that returns the **Factory2D** object which supplies the appropriate methods to create 2D geometric elements. Once you have finished creating 2D geometric elements,

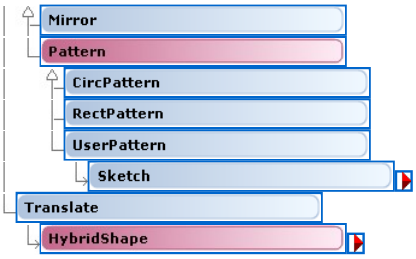
"close the sketch edition" using the **CloseEdition** method. These two methods correspond to the commands that let you interactively enter and leave the sketch. You can set constraints to the 2D geometric elements using the methods supplied by the **Constraints** collection object aggregated by the sketch.

# Shape Object Model Map

See Also Legend





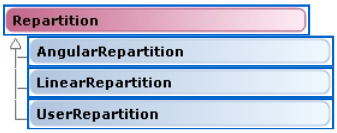




Shapes are classified into those built on a sketch, such as a pad or a hole, those which represent boolean operations, such as a split or an intersection, those which are used to dress-up sketch-based shapes, such as a fillet or a chamfer, and finally those which transform an original shape, transform meaning here duplicates, such as mirror and pattern.

Sketch-based shapes rely obviously on sketches, and the **Sketch** property sets or retrieves the sketch associated with the shape. The **Prism** abstract object federates properties and methods for the **Pad** and the **Pocket** objects.

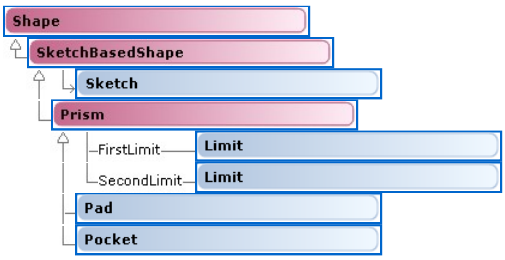
# Repartition Abstract Object

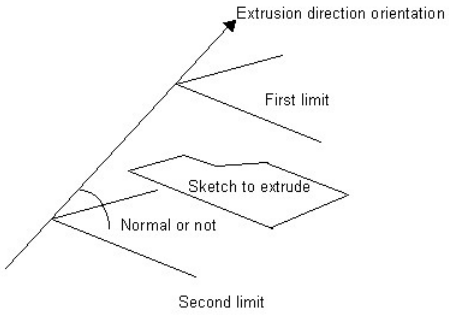
See Also Legend UseCases Properties Methods




# Prism Object

See Also Legend UseCases Properties Methods

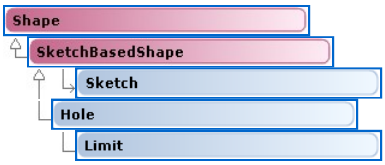


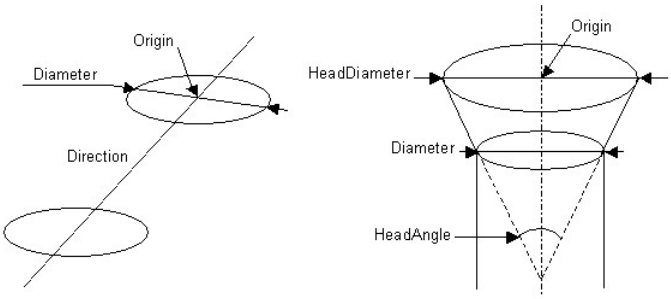




The **Prism** object provides to the **Pad** or to the **Pocket** objects the extrusion direction set or retrieved thanks to the **SetDirection** or **GetDirection** methods. This extrusion direction can be normal to the sketch plane or not. This is expressed using the **CatPrismExtrusionDirection** enumeration in the **DirectionType** property. The extrusion direction has an orientation expressed using the **CatPrismOrientation** enumeration in the **DirectionOrientation** property. The **Pad** or the **Pocket** object can be symmetric to the sketch plane. This is stored in the **IsSymmetric** property. Finally, the pad or the pocket has two limits stored in the **FirstLimit** and **SecondLimit** properties as **Limit** objects. The **Pad** and the **Pocket** objects do not hold specific properties or methods.

# Hole Object

See Also Legend UseCases Properties Methods






# Generating a Chain Automatically

This use case primarily focuses on the methodology to Generate automatically a hain.

---

Before you begin :

- You should first launch CATIA and open a new Part.
- Launch the following macro :

Where to find the macro : CAAPriGenChainSource.htm

This use case:

1. Gets the number of Chain Links
2. Creates the first two Chain Links
3. Creates the Construction line for a Rectangular Pattern
4. Creates the other Chain Links with a Rectangular Pattern
5. Updates the Part

1. **Gets the number of Chain Links**

   First, the UC displays a dialog box to set the number of Chain Link(s).

```
Dim nbChain Links As Integer
nbLinks = InputBox("Number of Chain Links :")
...
```

2. **Creates the first two Chain Links**

```
 ...
' Gets the First Link Sketch Plane
Dim originElements1 As OriginElements
Set originElements1 = part1.OriginElements

Dim planeLink1 As Reference
Set planeLink1 = originElements1.PlaneZX

' Gets the First Link Sketch Absolute Axis
Dim absoluteAxisSketch1(8)
absoluteAxisSketch1(0) = 0#
absoluteAxisSketch1(1) = 0#
absoluteAxisSketch1(2) = 0#
absoluteAxisSketch1(3) = 1#
absoluteAxisSketch1(4) = 0#
absoluteAxisSketch1(5) = 0#
absoluteAxisSketch1(6) = 0#
absoluteAxisSketch1(7) = 0#
absoluteAxisSketch1(8) = 1#

' Sets the First Link Anchor Point
Dim anchorPointLink1(1) As Double
anchorPointLink1(0) = 0
anchorPointLink1(1) = 0

' Creates the First Link
Dim shaft1 As Shaft
Set shaft1 = CreateLink(part1, body1, planeLink1, absoluteAxisSketch1, anchorPointLink1)

...
```

   The function CreateLink (annexe 2) creates a Chain Link from a Plane and an Anchor Point in this Plane. The Chain Link is created in the plane with the anchor p as its center.

   The second Chain Link is created the same way with the same function.

3. **Creates the Construction line for the Rectangular Pattern**

```
...
' Creates the 1st Point
Dim hybridShapeFactory1 As HybridShapeFactory
Set hybridShapeFactory1 = part1.HybridShapeFactory

Dim hybridShapePointCoord1 As HybridShapePointCoord
Set hybridShapePointCoord1 = hybridShapeFactory1.AddNewPointCoord(0#, 0#, 0#)
```

```
body1.InsertHybridShape hybridShapePointCoord1

part1.InWorkObject = hybridShapePointCoord1

Dim reference15 As Reference
Set reference15 = part1.CreateReferenceFromObject(hybridShapePointCoord1)

' Creates the 2nd Point
Dim hybridShapePointCoord2 As HybridShapePointCoord
Set hybridShapePointCoord2 = hybridShapeFactory1.AddNewPointCoord(0#, 0#, 135#)

body1.InsertHybridShape hybridShapePointCoord2

part1.InWorkObject = hybridShapePointCoord2

Dim reference16 As Reference
Set reference16 = part1.CreateReferenceFromObject(hybridShapePointCoord2)

' Creates the Line
Dim hybridShapeLinePtPt1 As HybridShapeLinePtPt
Set hybridShapeLinePtPt1 = hybridShapeFactory1.AddNewLinePtPt(reference15, reference16)

body1.InsertHybridShape hybridShapeLinePtPt1

part1.InWorkObject = hybridShapeLinePtPt1

...
```

4. **Creates the other Chain Links with a Rectangular Pattern**

```
...
If (nbLinks Mod 2 <> 0) Then
   CreateRectPattern part1, shaft1, hybridShapeLinePtPt1, (nbLinks \ 2) + 1
Else
   CreateRectPattern part1, shaft1, hybridShapeLinePtPt1, nbLinks \ 2
End If

CreateRectPattern part1, shaft2, hybridShapeLinePtPt1, nbLinks \ 2
...
```

The method CreateRectPattern (annexe 3) creates a Rectangular Pattern from a Bodypart, a Shaft, a Line and an Integer. It creates the Integer number of Shafts in direction of the Line, spaced by 265mm.

5. **Updates the Part**

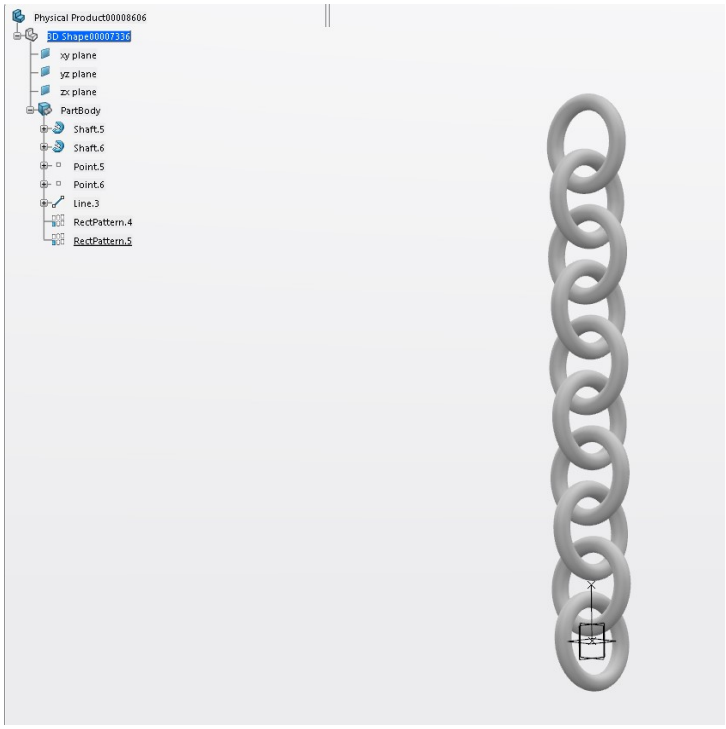
```
...
part1.Update
```

## In Short :

**This use case shows how to create sketches, Shafts, Hybrid Shapes and a Rectangular Patterns in Part Design.**

## Annexes :

**Annexe 1 : The Generated Chain**




**Annexe 2 : The Function CreateLink Table**

```
Function CreateLink(part1 As Part, body1 As Body, planeLink As Reference, absoluteAxisSketch() As Variant, anchorPoint() As Double
   ' Creates the Sketch
   Dim sketch1 ' As Sketch
   Set sketch1 = body1.Sketches.Add(planeLink)

   sketch1.SetAbsoluteAxisData absoluteAxisSketch

   ' Gets the 2D Factory
   Dim factory2D1 As Factory2D
   Set factory2D1 = sketch1.OpenEdition()

   ' Creates a Circle
   Dim point2D1 As Point2D
   Set point2D1 = factory2D1.CreatePoint(anchorPoint(0), 100 + anchorPoint(1))

   point2D1.ReportName = 1

   Dim circle2D1 As Circle2D
   Set circle2D1 = factory2D1.CreateClosedCircle(anchorPoint(0), 100 + anchorPoint(1), 20#)

   circle2D1.CenterPoint = point2D1

   circle2D1.ReportName = 2

   ' Creates the construction Line
   Dim point2D2 As Point2D
   Set point2D2 = factory2D1.CreatePoint(anchorPoint(0) - 100, anchorPoint(1))

   point2D2.ReportName = 3

   Dim point2D3 As Point2D
   Set point2D3 = factory2D1.CreatePoint(anchorPoint(0) + 100, anchorPoint(1))

   point2D3.ReportName = 4

   Dim line2D3 As Line2D
   Set line2D3 = factory2D1.CreateLine(anchorPoint(0) - 100, anchorPoint(1), anchorPoint(0) + 100, anchorPoint(1))

   line2D3.ReportName = 5

   line2D3.StartPoint = point2D2

   line2D3.EndPoint = point2D3

   sketch1.CenterLine = line2D3

   sketch1.CloseEdition

   part1.InWorkObject = sketch1

   ' Creates the Shaft
   Dim shapeFactory1 As ShapeFactory
   Set shapeFactory1 = part1.ShapeFactory

   Dim reference6 As Reference
   Set reference6 = part1.CreateReferenceFromName("")

   Dim shaft1 As Shaft
   Set shaft1 = shapeFactory1.AddNewShaftFromRef(reference6)

   ' Sets the Shaft Angle
```

```
    Dim angle1 As Angle
    Set angle1 = shaft1.FirstAngle

    angle1.Value = 360#

    ' Sets the Sketch on which the Shaft is based
    Dim reference7 As Reference
    Set reference7 = part1.CreateReferenceFromObject(sketch1)

    shaft1.SetProfileElement reference7

    ' Sets the Shaft Revolute Axis
    shaft1.RevoluteAxis = line2D3

    ' Returns the Shaft
    Set CreateLink = shaft1
End Function
```

**Annexe 3 : The method CreateRectPattern**

```
Sub CreateRectPattern(part1 As Part, shaft1 As Shaft, hybridShapeLinePtPt1 As HybridShapeLinePtPt, nbLinks As Integer)
    ' Gets the Shape Factory
    Dim shapeFactory1 As ShapeFactory
    Set shapeFactory1 = part1.ShapeFactory

    ' Creates the Rectangular Pattern
    Dim reference19 As Reference
    Set reference19 = part1.CreateReferenceFromName("")

    Dim reference20 As Reference
    Set reference20 = part1.CreateReferenceFromName("")

    Dim rectPattern1 As RectPattern
    Set rectPattern1 = shapeFactory1.AddNewRectPattern(Nothing, nbLinks, 1, 265#, 0#, 1, 1, reference19, reference20, True, True, (

    rectPattern1.FirstRectangularPatternParameters = catInstancesandSpacing

    rectPattern1.SecondRectangularPatternParameters = catInstancesandSpacing

    ' Sets the Rectangular Pattern first Direction
    Dim reference21 As Reference
    Set reference21 = part1.CreateReferenceFromObject(hybridShapeLinePtPt1)

    rectPattern1.SetFirstDirection reference21

    ' Sets the Item to Copy
    rectPattern1.ItemToCopy = shaft1
End Sub
```

# Generative Shape Design Object Model Map

See Also Legend


◆ Available with *Generative Shape Design* product only.
◆◆ Available with the *Generative Shape Optimizer* product only.

**HybridShape** objects are created using the **HybridShapeFactory** object aggregated by the **Part** object and are retrieved, from the Part object's **HybridBodies** and **HybridShapes** collections. Follow the ▲ arrow to find more information about those objects.

# Creating a Join Surface

This use case fundamentally illustrates creating a Join surface using the geometry created (Fill and Extrude).

Related Topics
*<topic1>*
*<topic2>*

Before you begin: Note that:

- Launch CATIA

Where to find the macro: CAAScdGsiUcCreateJoinSurfaceSource.htm

This use case can be divided in six steps:

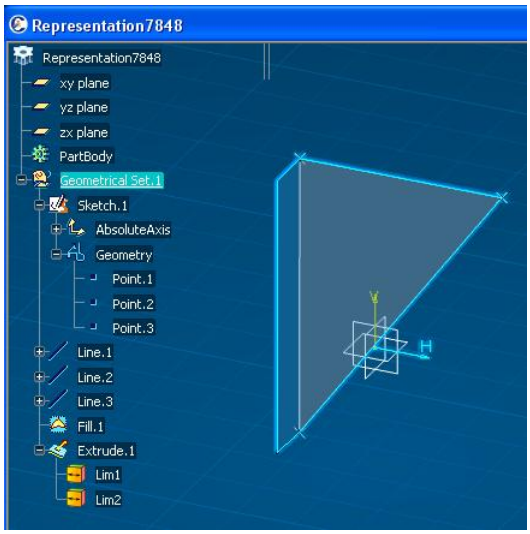
1. Creates 3DShape with Geometries
2. Retrieves Geometrical Set containing input Geometries
3. Creates Join Surface
4. Inserts Join in the Geometrical Set
5. Sets Join as Current Object
6. Updates Part Object

1. **Creates 3DShape with Geometries**

```
...
    Dim Part1 As Part
    CreateFillnExtrude Part1
...
```

The CreateFillnExtrude sub routine creates a new 3DShape with geometry as shown in the following picture.

Fig. 1 Creating Geometry with Fill and Extrude




`Part1` is the Part Object of the 3D Shape, the object aggregating all the objects making up the 3D shape, whose `Fill.1` and `Extrude.1` are used to create the join.

2. **Retrieves Geometrical Set containing input Geometries**

```
...
    Dim oHybridBodies As HybridBodies
    Set oHybridBodies =  Part1.HybridBodies

    Dim oHybridBody As HybridBody
    Set oHybridBody =  oHybridBodies.Item("Geometrical Set.1")
...
```

The first line retrieves the `HybridBodies` collection object. From this collection, `oHybridBodies`, we further retrieve the `HybridBody` object whose external name is "`Geometrical Set.1`".

3. **Creates Join Surface**

The Join surface is created between two surfaces, `Fill.1` and `Extrude.1`, contained inside `Geometrical Set.1` retrieved in the previous step. Once each input is retrieved, they are converted into `Reference` object. This operation is required in order to use the `Reference` objects as input for the Join operation.

i. Retrieves Fill.1, and creates a reference for it

```
...
    Dim oHybridShapeFill1 As HybridShape
    Set oHybridShapeFill1 = oHybridBody.HybridShapes.Item("Fill.1")

    Dim  Reference1 As Reference
    Set Reference1 = Part1.CreateReferenceFromObject(oHybridShapeFill1)
...
```

Reference1 is an object Reference of `Fill.1` retrieved in the `Geometrical Set.1` represented by `HybridBody` object.

ii. Retrieves Extrude.1 and creates a reference for it

```
...
    Dim oHybridShapeExtrude1 As HybridShape
    Set oHybridShapeExtrude1 = oHybridBody.HybridShapes.Item("Extrude.1")

    Dim  Reference2 As Reference
    Set Reference2 = Part1.CreateReferenceFromObject(oHybridShapeExtrude1)
...
```

Reference2 is an object Reference of `Extrude.1` retrieved in the `Geometrical Set.1` represented by `oHybridBody` object.

iii. Creates a Join (also named assemble) between Fill.1 and Extrude.1

```
...
    Dim oHybridShapeFactory As Factory
    Set oHybridShapeFactory = Part1.HybridShapeFactory

    Dim oJoin.1 As HybridShapeAssemble
    Set oJoin.1 = oHybridShapeFactory.AddNewJoin(Reference1, Reference2)
...
```

The join object is created thanks to a `Factory` object, that you retrieve from the `Part` object. Its **HybridShapeFactory** property retrieves the hybrid shape factory, named here `oHybridShapeFactory`. The join object, `oJoin.1`, is created using the reference objects of both surfaces, retrieved in the above steps to join.

4. **Inserts Join in the Geometrical Set**

Once created the join object must be aggregated in the Part object, in other words aggregated to one of its own set object.

```
...
    oHybridBody.AppendHybridShape oJoin.1
...
```

The newly created join, `oJoin.1`, [Fig.2] is inserted inside "Geometrical Set.1" represented by `oHybridBody` object ( See step [2] ) using the

**AppendHybridShape** method.

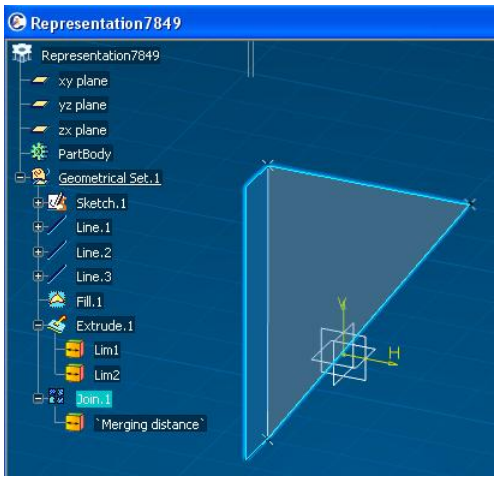
5. **Sets Join as Current Object**

   Sets the join, `oJoin.1`, as the in work object of the Part object.

```
...
    Part1.InWorkObject = oJoin.1
...
```

   Just below you can see Join.1 is highlighted.

   Fig. 2 Join Selected as Current Object



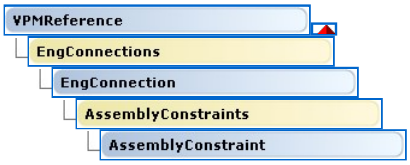

6. **Updates Part Object**

```
...
    Part1.Update
...
```

   Updates the Part Object, `Part1`, result with respect to its specifications.

# Assembly Design Object Model Map

See Also Legend




# Retrieving the Engineering Connection from the Current Product Structure

This use case primarily focuses on the methodology to retrieve Engineering Connections from a Product Structure.

---

Related Topics
*<topic1>*
*<topic2>*

Before you begin: Note that:

- You should first launch CATIA and open or create an existing product structure containing Engineering Connection.

Where to find the macro: CAAScdAssUcRetrieveMCXSource.htm

This use case can be divided in four steps:

1. Retrieves the current Editor from the product Structure
2. Retrieves the Product Structure Root
3. Retrieves the collection of the Engineering Connections
4. Loop on the Engineering Connections and retrieves the name of the current Engineering Connection

1. **Retrieves the current Editor from the product Structure**

   As a first step, a product Structure containing Engineering Connections must be openned in the CATIA session.

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

2. **Gets the Product Reference Root from the Editor**

   Hereunder, the usual code to retrieve the VPMReference from the current editor.

```
 ...
 Dim MyPrdService   As  PLMProductService
 Set MyPrdService = myEditor.GetService("PLMProductService")
 Dim myRootOccurrence As  VPMRootOccurrence
 Set myRootOccurrence = MyPrdService.RootOccurrence
 Set myMCXParent = myRootOccurrence.ReferenceRootOccurrenceOf
```

3. **Gets the Collection of Engineering Connections from Product Reference**

   `CATEngConnections` keyword allowing to get the collection of Engineering Connection from the Product Reference Root.

```
...
 Dim myEngConnections As  EngConnections
 Set myEngConnections = myMCXParent.GetItem("CATEngConnections")
...
```

4. **Loop on the Engineering Connections and gets the Engineering Connection names**

```
...
 Dim myEngConnection As  EngConnection
 Dim i  As  Integer
 For i = 1 To myEngConnections.Count
   Set myEngConnection = myEngConnections.Item(i)
   MsgBox " my Engineering Connection name = " & myEngConnection.Name
 Next
End Sub
```

# Creating an Assembly Constraint Between Two Instances of a Basic Pad

This use case primarily focuses on the methodology to create an Assembly Constraint between two instances of a basic pad.

---

Related Topics
*<topic1>*
*<topic2>*

Before you begin: Note that:

- You should first launch CATIA and import the `EngCntRootProduct.3dxml` file supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdAssDesign\samples\` where `InstallRootFolder` is the directory where the CAA CD-ROM is installed.

Where to find the macro: CAAScdAssUcCstCreationSource.htm

This use case can be divided in seven steps:

1. Searches and opens model which is named as "EngCntRootProduct"
2. Retrieves the Root Product
3. Retrieves the collection of the Engineering Connection
4. Creates an Engineering Connection between the two instances aggregated by the Root Product
5. Retrieves the collection of the Assembly Constraints
6. Creates a distance "plan-plan" constraint between faces identified by their publication
7. Sets the driving mode
8. Adds the Engineering Connection in the Selection Object to display it

1. **Searches and opens model which is named as "EngCntRootProduct"**

   As a first step, the UC retrieves a model "EngCntRootProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim EngCntEditor As  Editor
 OpenProduct EngCntEditor
...
```

   The function **`OpenProduct`** returns **`EngCntEditor`**, a Editor object. After searching and opening of EngCntRootProduct model from underlying database the current active editor is returned in **`EngCntEditor`**.

2. **Retrieves the VPMReference from the current Editor**

   As a next step, the UC retrieves the VPMReference object in EngCntRootProduct product

```
 ...
Dim MyPrdService   'As  PLMProductService
Set MyPrdService = EngCntEditor.GetService("PLMProductService")
Dim myRootOccurrence 'As  VPMRootOccurrence
Set myRootOccurrence = MyPrdService.RootOccurrence
Dim myMCXParent 'As  VPMReference
Set myMCXParent = myRootOccurrence.ReferenceRootOccurrenceOf
 ...
```

3. **3- Retrieves the collection of Engineering Connection from a Product Root**

   In this step UC retrieves the collection of the Engineering Connections.

```
...
Dim myEngConnections 'As  EngConnections
Set myEngConnections = myMCXParent.GetItem("CATEngConnections")
...
```

4. **4- Creates an Engineering Connection between the two instances aggregated by the Product Root**

   In this step UC creates a new User Defined Engineering Connection between the two following impacted: Part1.1 and Part1.2.

   These instances are identified by the their chain of characters in the product Structure

```
...
ReDim  myImpacteds(1) 'As  Collection
myImpacteds(0) = "Part1.1"
myImpacteds(1) = "Part1.2"
Set myNewMCX = myEngConnections.Add(catUserDefined, myImpacteds)
...
```

5. **5- Retrieves the collection of Assembly Constraints.**

   In this step UC retrieves the collection of Assembly Constraints for a given Engineering Connection.

```
...
Dim myConstraints 'As  AssemblyConstraints
Set myConstraints = myNewMCX.AssemblyConstraints
...
```

6. **6- Creates a distance constraint between 2 planar faces identified by their publication**

   In this step UC creates an Assembly constraint between 2 faces identified by their publication.

   The publication are identified by their name.

```
...
ReDim  myGeometries(1) 'As  Collection
myGeometries(0) = "FacePart1.1"
myGeometries(1) = "FacePart1.2"
Dim myNewAssemblyConstraint11 'As   AssemblyConstraint
Set myNewAssemblyConstraint11 = myConstraints.Add(catDistancePlanePlane, myGeometries)
...
```

7. **7- Sets the Driving Mode for the created constraint**

   In this step UC, the constraint is set to DrivingMode and the distance between the two faces is set to 200mm.

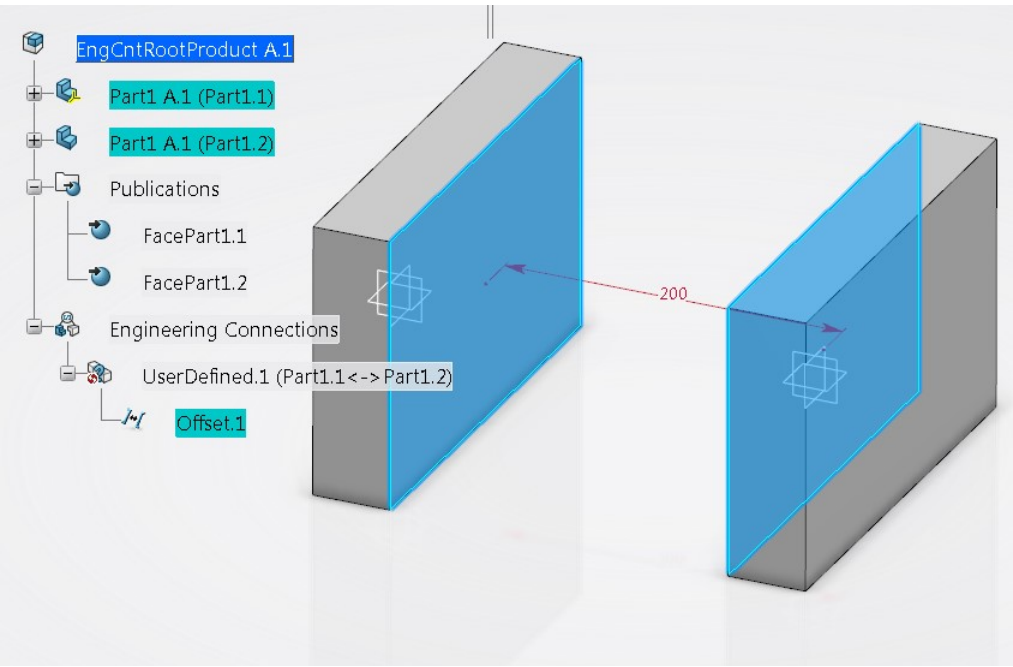
```
...
myNewAssemblyConstraint11.Mode = catDrivingMode
myNewAssemblyConstraint11.SetValue 1, 200.0
...
```

8. **8- Adds the Engineering Connection in the Selection Object to display it**

   The display of Engineering Connections is managed by settings, by default it displayed only if the Engineering Connection is selected.

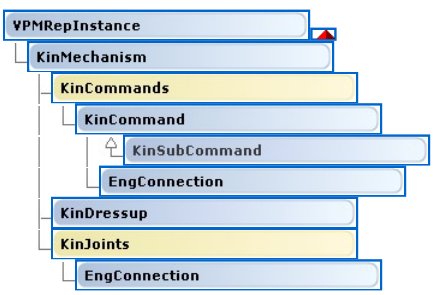
```
...
CATIA.ActiveEditor.Selection.Add myNewAssemblyConstraint11
...
```

Fig.1: Member




# Mechanical Systems Design / Experience Object Model Map

See Also Legend




- Use the **GetItem** method to retrieve the **KinMechanism** object from a **VPMRepInstance** object, assuming `oRepRef` is a **VPMRepInstance** object:

```
Dim oMechRep As KinMechanism
Set oMechRep = oRepRef.GetItem("MECHRep")
```

- Use the **GetItem** to retrieve the **KinDressup** object from a **KinMechanism** object, assuming `oMechRep` is the **KinMechanism** object:

```
Dim oDressup As KinDressup
Set oDressup = oMechRep.GetItem("KinDressup")
```

- Use the **GetItem** to retrieve the **KinSubCommand** object from a **KinCommand** object, assuming MyCommand is the **KinCommand** object:

```
Dim SubCmd As KinSubCommand
Set SubCmd= MyCommand.GetItem("KinSubCommandObj")
```

# Scanning a Mechanism

Related Topics
*<topic1>*
*<topic2>*

Before you begin: Note that:

- You should first launch CATIA and import the `Moteur.3dxml` file supplied in folder `InstallRootFolder\CAADoc\Doc\CAAScdKin\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdKinScanSource.htm

This use case can be divided in six steps:

1. Retrieves the current Editor
2. Retrieves the Product Service from the Editor
3. Gets the discipline of all representation of the tree
4. Gets the Mechanism Representation if the V_discipline is the good one
5. Scans Joints under the Mechanism Representation
6. Scans Commands and retrieves some information

1. **Retrieves the current Editor**

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

2. **Retrieves the Product Service from the Editor**

```
...
            Dim MyPrdService   As  PLMProductService
            Set MyPrdService = myEditor.GetService("PLMProductService")
            Dim myRootOccurrence As  VPMRootOccurrence
            Set myRootOccurrence = MyPrdService.RootOccurrence
...
```

3. **Gets the discipline of all representation of the tree**

   In this step, we search through all the reps of the tree.

```
...
            For Each MyEntity In MyPrdService.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")
...
```

4. **Gets the Mechanism Representation if the V_discipline is the good one**

   We sort on the discipline of all reps to find mechanisms.

   Discipline for mechanisms is Mechanism for new products and Simulation for old ones.

```
...
             If  ( attr = "Mechanism" ) Then
               Dim MyMechRep  As  KinMechanism
               Set MyMechRep = MyRepRef.GetItem("MECHRep")
...
```

5. **Scans Joints under the Mechanism Representation.**

   In this step, we get the joints collection.

```
...
               Dim MyJoints  As  KinJoints
               Set MyJoints = MyMechRep.Joints
...
```

   For each engineering connection we get some information:

```
...
              For  i = 1 To MyJoints.Count Step 1
                Dim MyJoint As EngConnection
                Set MyJoint = MyJoints.Item(i)
...
```

   As the Name

```
...
```

```
                MsgBox " my Engineering Connection name = " & MyJoint.Name
...
```

Or the type

```
...
                MsgBox " my Engineering Connection Type = " & MyJoint.Type
...
```

6. **Scans Commands and retrieves some information.**

   In this step, we get the commands collection.

```
...
                Dim MyCommands  As  KinCommands
                Set MyCommands = MyMechRep.Commands
...
```

   For each engineering connection we get some information:

```
...
              For  i = 1 To MyCommands.Count Step 1
                Dim MyCommand As KinCommand
                Set MyCommand = MyCommands.Item(i)
...
```

   As the Name

```
...
                MsgBox " Command name = " & MyCommand.Name
...
```

   Or the current value

```
...
                MsgBox " Current Value = " & MyCommand.CurrentValue
...
```

# Simulating a Mechanism

Before you begin: Note that:

- You should first launch CATIA and import the `Moteur.3dxml` file supplied in folder `InstallRootFolder\CAADoc\Doc\CAAScdKin\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Related Topics
*<topic1>*
*<topic2>*

Where to find the macro: CAAScdKinRunSource.htm

This use case can be divided in seven steps:

1. Retrieves the current Editor
2. Retrieves the Product Service from the Editor
3. Gets the discipline of all representation of the tree
4. Gets the Mechanism Representation if the V_discipline is the good one
5. Prepares the Solver
6. Runs the all Mechanism
7. Runs an unique command
8. Cleans the Solver

1. **Retrieves the current Editor**

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

2. **Retrieves the Product Service from the Editor**

```
...
          Dim MyPrdService   As  PLMProductService
          Set MyPrdService = myEditor.GetService("PLMProductService")
...
```

3. **3- Gets the discipline of all representation of the tree**

   In this step, we search through all teh reps of the tree.

```
...
          For Each MyEntity In MyPrdService.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")
...
```

4. **4- Gets the Mechanism Representation if the V_discipline is the good one**

   We sort on the discipline of all reps to find mechanisms.

   Discipline for mechanisms is Mechanism for new products and Simulation for old ones.

```
...
```

```
                If ( attr = "Mechanism" ) Then
                Dim MyMechRep  'As  KinMechanism
                Set MyMechRep = MyRepRef.GetItem("MECHRep")
...
```

5. **5- Prepares the Solver.**

   This step is mandatory before simulate the Mechanism. Otherwise the macro will be return an error.

```
...
                MyMechRep.PrepareSimulation
...
```

6. **6- Runs the all Mechanism.**

   The array in entries must have the same dimension as the number of kinematics commands.

```
...
                Dim MyValues(4)  'As CATSafeArrayVariant
                MyValues(0)=150.0
                MyValues(1)=125.0
                MyValues(2)=45.0
                MyValues(3)=-17.0
                MyValues(4)=-17.0
...
```

   Then you can run your mechanism for any number of steps.

```
...
                For i = 0 To 10 Step 1
                  MyMechRep.RunSimulation(MyValues)
                  MyValues(0)=MyValues(0)+5.0
                  MyValues(1)=MyValues(1)+5.0
                  MyValues(2)=MyValues(2)+5.0
                  MyValues(3)=MyValues(3)+1.5
                  MyValues(4)=MyValues(4)+1.5
                Next
...
```

7. **7- Runs an unique command.**

   Despite simulate the all mechanism you have the choice to simulate an unique command.

```
...
                Dim val As Double
                val = MyValues(1)
                For i = 0 To 10 Step 1
                  val = val - 5.0
                  MyMechRep.RunCommand  "Command.2" , val
...
```

8. **8- Cleans the Solver.**

   The step of cleaning has to be done after all simulations are finished.
   And another step of prepare cannot be done before this step.
   Beetween a couple of Prepare/Clean, any number of simulation steps can be done

```
...
                MyMechRep.CleanSimulation
...
```

# Simulating a mechanism with subcommands

Before you begin:

- Open the 3DEPERIENCE
  Import the SliderCrankMechanism.3dxml file supplied in folder
  `InstallRootFolder\CAADoc\Doc\CAAScdKin\samples\` where `InstallRootFolder` is the folder
- where the CAA CD-ROM is installed.

For more information on the macro, see : CAAScdKinRunSubMechSource.htm

This use case is divided in 8 steps.

1 Gets the current Editor

2 Gets the Product Service from the Editor

3 Identify the mechanism

4 Scan all kinematics commands

5 Prepares the Solver

6 Run a first simulation by giving all the commands values

7 Run a second simulation by giving only a target value

8 Clean the simulation

Steps 1 to 3 have already been described in previous use cases. Only steps 4 to 8 will be be described in this use case.

4. **Scan all kinematics commands**

   In this step, all the kinematics commands are retrieved.

To get a KinSubCommand object from a KinCommand object, use the GetItem method. The SubMechanism and SubMechanismProductOccurrence properties allow you to identify the location of the subcommand. Those properties will appear empty if the kinematics command is directly under the mechanism.

```
Dim MyCommands As KinCommands
Set MyCommands = MyMechRep.Commands

Dim i As Integer
For i = 1 To MyCommands.Count
  Dim MyCommand As KinCommand
  Set MyCommand = MyCommands.Item(i)
  Msg = " Command : " & MyCommand.Name
  MsgBox (Msg)

  Dim SubCmd As KinSubCommand
  Set SubCmd = MyCommand.GetItem("KinSubCommandObj")

  Dim SubMech As KinMechanism
  Set SubMech = SubCmd.SubMechanism
  If Not SubMech Is Nothing Then
    MsgBox ("This command is owned by the sub mechanism, " & SubMech.Name)
  End If

  Dim SubMechOcc As VPMOccurrence
  Set SubMechOcc = SubCmd.SubMechanismProductOccurrence
  If Not SubMechOcc Is Nothing Then
    MsgBox ("This sub mechanism is defined in " & SubMechOcc.Name)
  End If
Next
```

The current values of all the kinematics commands are saved.

```
Dim MyValues(2)
MyValues(0) = MyCommands.Item(1).CurrentValue
MyValues(1) = MyCommands.Item(2).CurrentValue
MyValues(2) = MyCommands.Item(3).CurrentValue
Msg = " Current command values are "
Msg = Msg & MyValues(0) & ", " & MyValues(1) & ", " & MyValues(2)
MsgBox (Msg)
```

5. **Prepare the Solver**

   The solver is initialized for the kinematics simulation.

   ```
   MyMechRep.PrepareSimulation
   MsgBox ("PrepareSimulation is done")
   ```

6. **Run a first simulation with all command values**

   MyValues is the target that the solver must reach for the kinematics simulation. This range includes 3 values, for each kinematics command.

   ```
   MyValues(0) = MyValues(0) + 30#
   MyValues(1) = MyValues(1) + 30#
   MyValues(2) = MyValues(2) + 30#

   Call MyMechRep.RunSimulation(MyValues)
   MsgBox ("A first kinematic simulation is done")

   Dim V(2) As Double
   V(0) = MyCommands.Item(1).CurrentValue
   V(1) = MyCommands.Item(2).CurrentValue
   V(2) = MyCommands.Item(3).CurrentValue
   Msg = "The mechanism has reached the following command values : "
   Msg = Msg & V(0) & ", " & V(1) & ", " & V(2)
   MsgBox (Msg)
   ```

7. **Run a second simulation with a single command value**

   In this second simulation, only the third command value is targeted.

   ```
   Dim val As Double
   val = V(2)
   val = val + 30#

   Call MyMechRep.RunCommand(2, val)
   MsgBox ("A second kinematic simulation is done")

   V(0) = MyCommands.Item(1).CurrentValue
   V(1) = MyCommands.Item(2).CurrentValue
   V(2) = MyCommands.Item(3).CurrentValue
   Msg = "The mechanism has reached the following command values : "
   Msg = Msg & V(0) & ", " & V(1) & ", " & V(2)
   MsgBox (Msg)
   ```

   You can go through steps 6 and 7 again to run as many simulations as necessary before cleaning the simulation.

8. **Clean the simulation**

   In this last step, the solver is cleaned.

   ```
   MyMechRep.CleanSimulation
   MsgBox ("CleanSimulation is done")
   ```

# Editing the Mechanism Dressup

Before you begin:

- Launch 3DEXPERIENCE and import the `SliderCrankMechanism.3dxml` file located in `InstallRootFolder\CAADoc\Doc\CAAScdKin\samples\`

`InstallRootFolder` where InstallRootFolder is the directory where the CAA CD-ROM is installed.

Where to find the macro: CAAScdKinDressupSource.htm

This sample shows how to use the **GetDressupProducts, AttachDressupProduct , DetachDressupProduct** methods, and the **MechanismProducts** property.

Open **SliderCrankMechanism** product before launching the macro. It contains a mechanism pointing to the Part3.1 product. The macro will list all the products attached to Part3.1. Only Product1.2 is attached to it. Each attached product will be detached from Part3.1. Then, another Product, Product1.1, will be attached to Part3.1. In the last step, all the mechanism products will be listed.

This sample is divided in 13 steps:

1. 1 Gets the current editor
2. 2 Gets the product service from the Editor
3. 3 Gets all the occurrences under the root occurrence
4. 4 Gets the context
5. 5 Gets the discipline of all representation of the tree
6. 6 Gets the mechanism representation if the V_discipline is mechanism
7. 7 Gets Part3.1, an occurrence of the mechanism
8. 8 Gets all the occurrences attached to Part3.1
9. 9 Detaches all the products previously attached to Part3.1
10. 10 Gets Product1.1 VPMOccurrence
11. 11 Attaches Product1.1 to Part3.1
12. 12 Lists all the products attached to Part3.1
13. 13 Lists all the mechanism products

1. **Gets the Mechanism Representation**

```
...
            ' 6 Gets the Mechanism Representation if the V_discipline is Mechanism
            If (attr = "Mechanism") Then
            Dim MyMechRep As KinMechanism
            Set MyMechRep = MyRepRef.GetItem("MECHRep")
            MsgBox " The mechanism name is " & MyMechRep.Name
...
```

2. **Gets all the occurrences attached to Part3.1**

   Part3.1 is a mechanism product. With the **GetDressupProducts** method, all the dressup products attached to Part3.1 are obtained.

```
...
            ' 8 Gets all the occurrences attached to Part3.1
            Dim ListAttached
            ListAttached = MyMechRep.GetDressupProducts(Part31)
            MsgBox " call of GetDressupProducts "
...
```

3. **Detaches all the products previously attached to Part3.1**

   The **DetachDressupProduct** method detaches all the dressup products attached to Part3.1. The corresponding rigid connection will be also removed.

```
...
          ' 9 Detaches all the products previously attached to Part3.1
          Dim Maxi
          Maxi = UBound(ListAttached)
          MsgBox " Number of dressup attached product to Part3.1 = " & (Maxi + 1)
          Dim j As Integer
          For j = 0 To Maxi
             Dim Product As VPMOccurrence
             Set Product = ListAttached(j)
             Call MyMechRep.DetachDressupProduct(Product)
             MsgBox Product.Name & " is detached from " & Part31.Name
          Next
...
```

4. **Attaches Product1.1 to Part3.1**

   The **AttachDressupProduct** method creates a dressup link and a rigid engineering connection between Product1.1 and Part3.1.

```
...
          ' 11 Attaches Product1.1 to Part3.1
          Call MyMechRep.AttachDressupProduct(Part31, Product11)
          MsgBox " Attach Product1.1 to Part3.1"
...
```

5. **13 Lists all the mechanism products**

   The **MechanismProducts** property gives only the mechanism products and not the attached dressup products.

```
...
          ' 13 Lists all the mechanism products
          Dim MechProducts
          MechProducts = MyMechRep.MechanismProducts

          Dim Msg As String
          Dim i As Integer
          For i = 0 To UBound(MechProducts)
             Dim MecProduct As VPMOccurrence
             Set MecProduct = MechProducts(i)
             Msg = Msg & MecProduct.Name
          Next
          MsgBox " The mechanism contains the following product : " & Msg
...
```

# Overview

Technical Article

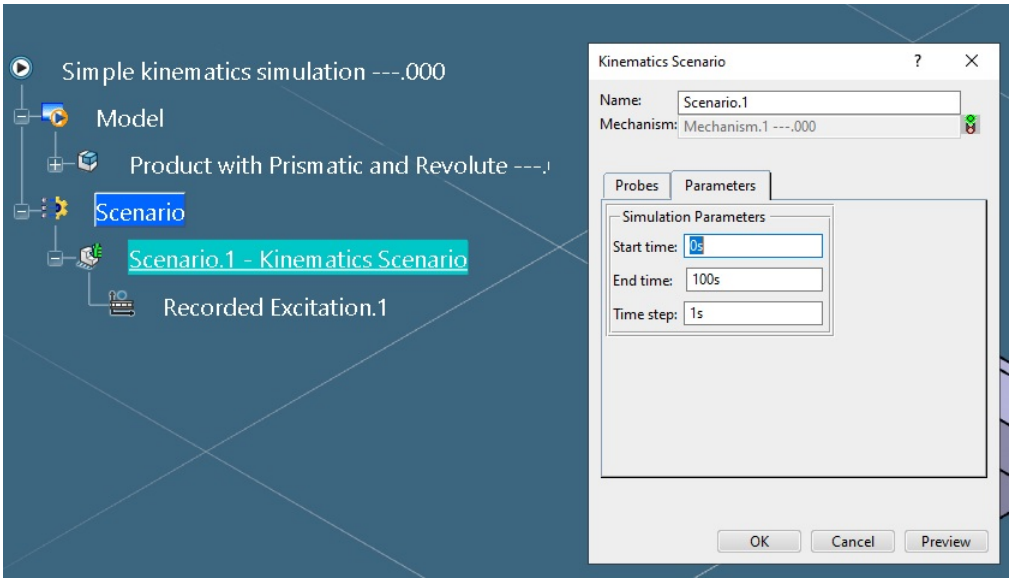
**Abstract**

This article presents an overview of a kinematics simulation with a recorded excitation



## Overview

Starting with a kinematics simulation pointing a product with a mechanism, a set of automation objects (**KinScenarioSpec**,**KinRecordedExcitation**) will allow to create and handle a kinematics scenario with a recorded excitation.




## APIs

### KinScenarioSpec

This object identifies a kinematics scenario. It allows to get/set the parameters of a kinematics scenario. The initialization of the link to the mechanism is done thru the **SimScenarioSpec** object.

APIs capabilities can be summarized as follows:

- Get the mechanism.
- Get/Set the Start time. 0s is the default value.
- Get/Set End time. 10s is the default value.
- Get/Set Time step. 1s is the default value.
- Add a recorded excitation. Only one recorded excitation can be created.
- Get a recorded excitation.
- Remove a recorded excitation.

The defaults values are the same as during an interactive 3D Experience session, respecting the following condition:

$0s \leq \text{Start time} \leq \text{End time}$

$0s \leq \text{Time step} \leq \text{End time} - \text{Start time}$

### KinRecordedExcitation

This object inherits from the **SimExcitation** object. It allows to get/set the parameters of the recorded excitation. This excitation can be seen as a time key frames plus a set of key frames values associated for each kinematics command of the mechanism.

| Time Key Frame | Key Frame for 1rst kinematics command | Key Frame for 2nd kinematics command |
|---|---|---|
| 0 s | 100 mm | 50 deg |
| 2 s | 110 mm | 60 deg |
| 4 s | 120 mm | 90 deg |
| 10 s | 150 mm | 100 deg |

APIs capabilities can be summarized as follows:

- Get/Set the key frame values for the time.
  The values of the key frames times should follow an ascending chronological order. The first time and the last time will always be set as the start time and the end time of the scenario.
- Get the size of the key frames.
- Get/Set a key frames values for a kinematics command.

During the creation of the recorded excitation, the key frame times should be defined the first and then the others key frames. The size of each the key frames must be the same and equal to size of the key frames times.

Setting a new time key frames times will reset the others key frames values automatically.

The scenario execution follows the time line and simulated the mechanism for each time step.

If a step does not appear in the time key frame, the missing time value will be linearly interpolated between the two closest time values. That means that giving just 2 values min and max for a kinematics command will allow to simulate all the intermediate values between min and max values.

| Time Key Frame | Key Frame for 1rst kinematics command |
|---|---|
| 0 s | 100 mm |
| 50 s | 150 mm |

In the above example, assuming that the step of the scenario is 1s. The kinematics simulation will be done by increasing the command value by 1mm from 100 mm up to 150 mm.

It is noticeable that a way to improve the accuracy of the simulation is to decrease the time step value, without changing the recorded excitation.

## References

[1] **SimScenarioSpec , SimExcitation**

## History

Version: **1** [Nov 2021] Document created

# Create a Kinematics Recorded Excitation

Before you begin: Note that:

- You should first launch CATIA and import the `Simple_kinematics_simulation.3dxml` file supplied in folder `InstallRootFolder\CAADoc\Doc\CAAScdKin\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdKinCreateRecExcitationSource.htm

Related Topics
*<topic1>*
*<topic2>*

This use case can be divided in the following steps:

1. Gets a link to the mechanism
2. Creates a kinematics scenario
3. Creates a recorded excitation
4. Populates the recorded excitation

Before launching the script be sure to have the simulation object opened in a 3D window.

1. **Gets a link to the mechanism**

   we create a link to the mechanism with the VPMRootOccurrence and the VPMRepInstance of the mechanism.

```
' Gets the root occurrence of the simulation
Dim oRootOcc As VPMRootOccurrence
Set oRootOcc = oSimulationService.GetProductRootOccurrence(oSimulation)

' Gets the rep instance of the mechanism
Dim oMechRepInst As VPMRepInstance
Set oMechRepInst = oRootOcc.RepOccurrences.Item(1).RelatedRepInstance

' Gets the mechanism object
Dim oMechRoot As KinMechanism
Set oMechRoot = oMechRepInst.ReferenceInstanceOf.GetItem("MECHRep")

Dim oBehaviorLink 'As AnyObject
Set oBehaviorLink = oProdService.ComposeLink(oRootOcc, oMechRepInst, oMechRoot)
ReDim LinkArray(0) 'As Variant
Set LinkArray(0) = oBehaviorLink
```

2. **Creates a kinematics scenario**

   We create a the kinematics scenario by giving the key word CATKinSolverOption and the link to the mechanism.

```
...
' Retrieves the SimulationSpecifications
Dim oSimSpecs As SimulationSpecifications
Set oSimSpecs = oSimulation.Specifications

' Retrieve the SimulationRepresentation
Dim oSimSpecRep As SimulationRepresentation
Set oSimSpecRep = oSimSpecs.Item(1)
```

```
' Retrieves the SimSpecsRepManager
Dim oScenarioRepManager As SimSpecsRepManager
Set oScenarioRepManager = oSimSpecRep.Root

' Retrieves the SimScenarioSpecs
Dim oScenarioSpecs 'As SimScenarioSpecs
Set oScenarioSpecs = oScenarioRepManager.ScenarioSpecs

' Creates a scenario
Set oScenarioSpec = oScenarioSpecs.Add("CATKinSolverOption", "", "", LinkArray).
```

3. **Creates a recorded excitation**

   We create the recored excitation attached to the kinematics scenatio thru the AddRecordedExcitation method.

```
...
'Gets the KinScenarioSpec
Dim oKinScenarioSpec 'As KinScenarioSpec
Set oKinScenarioSpec = oScenarioSpec.GetItem("KinScenario")

' Creates a recorded excitation under the kin scenario
Dim oKinExcitation 'As KinRecordedExcitation
Set oKinExcitation = oKinScenarioSpec.AddRecordedExcitation
.
```

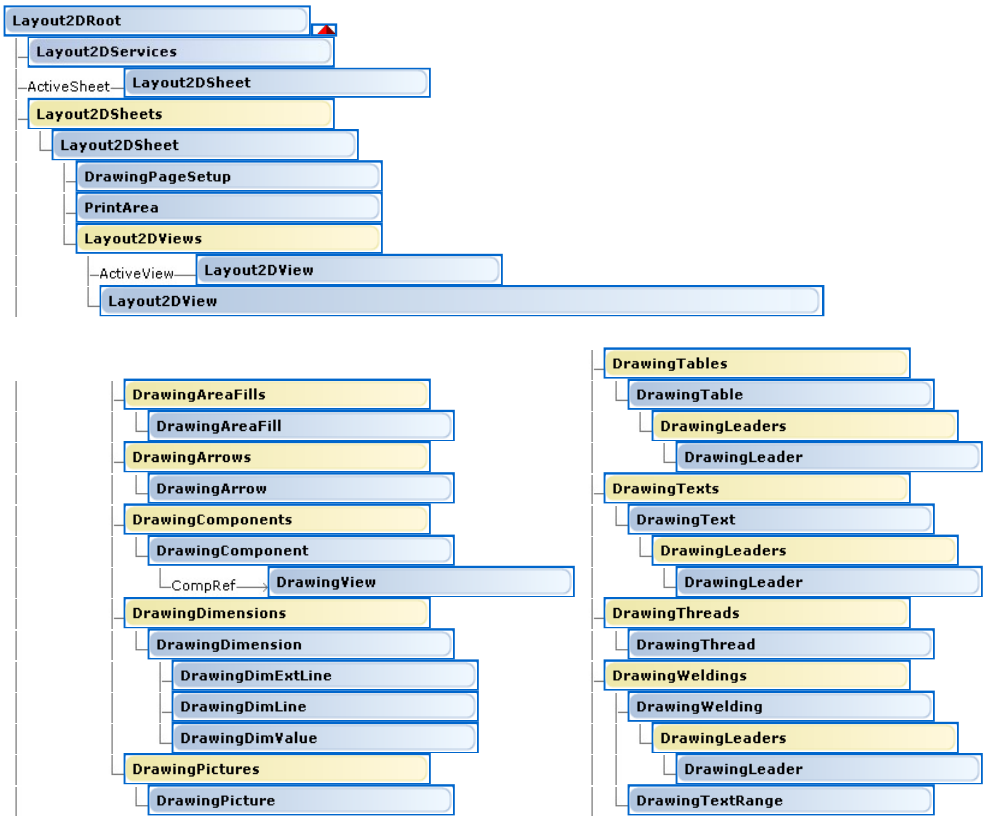
4. **Populates the recorded excitation**

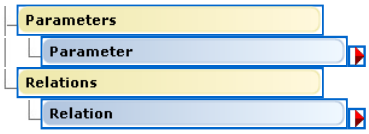
   We set the key frames for time with (0s, 100s), the 1rst command (0mm,100mm) and for the 2nd command (90deg,450deg).

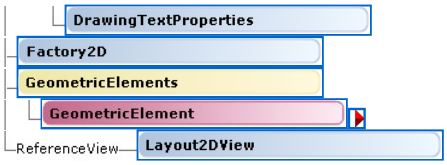
```
...
' Sets the key frames times
Dim KeyTimes(1) 'As CATSafeArrayVariant
KeyTimes(0) = 0#
KeyTimes(1) = 100#
oKinExcitation.SetKeyFramesTimes (KeyTimes)

Dim StartTime, EndTime, TimeStep  As Double
StartTime = 0#
EndTime = 100#
TimeStep = 1#

' Sets the times parameters
Call oKinScenarioSpec.SetTimeParameters(StartTime, EndTime, TimeStep)

Dim KeyCmdValues(1) 'As CATSafeArrayVariant

' Sets the key frames values for the 1rst kineatics command
KeyCmdValues(0) = 0#
KeyCmdValues(1) = 100#
oKinExcitation.SetKeyFramesValues MyCommand1, KeyCmdValues

' Set the key frames values for the 2nd kinematics command
KeyCmdValues(0) = 90#
KeyCmdValues(1) = 450#
oKinExcitation.SetKeyFramesValues MyCommand2, KeyCmdValues
```

# 2D Layout for 3D Design Object Model Map

See Also Legend

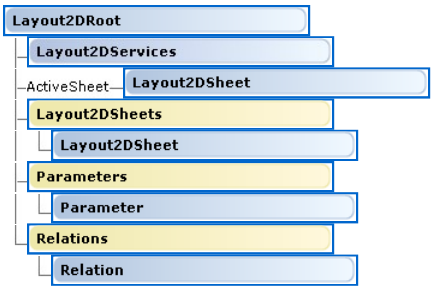





# Layout2DRoot Object

See Also Legend Use Cases Properties Methods




The **Layout2DRoot** object is the root object for 2D Layout applications. It is retrieved from the **Part** root object thanks to the **GetItem** method. Assume a 3D Part is active. If no **Layout2DRoot** object exists, you can create it thanks to the **Layout2DFactory** object, as follows:

```
Dim oPart As Part
Set oPart = CATIA.ActiveEditor.ActiveObject
Dim oLayRoot As Layout2DRoot
Set oLayRoot = oPart.GetItem("CATLayoutRoot")
If oLayRoot Is Nothing Then
   Dim oLayRootFact As Layout2DFactory
   Set oLayRootFact = oPart .GetItem("CATLayoutRootFactory")
   Set oLayRoot = oLayRootFact.Create2DLayout("ISO_3D")
End If
```

The **Layout2DRoot** object aggregates a **Layout2DSheets** collection object that contains **Layout2DSheet** objects, one of them being the active sheet. Each **Layout2DSheet** object contains a **Layout2DViews** collection object. This collection contains **Layout2DView** objects, one of them being the active view.

```
Dim cLaySheets As Layout2DSheets
Set cLaySheets = oLayRoot.Sheets

Dim oLaySheet1, oLaySheet2, oLaySheet3 As Layout2DSheet
Set oLaySheet1 = oLayRoot.ActiveSheet       'Retrieves the active sheet from the 2D layout root object
Set oLaySheet2 = cLaySheets.ActiveSheet     'Retrieves the active sheet from the sheet collection object
Set oLaySheet3 = cLaySheets.Item("Sheet.2") 'Retrieves a sheet using its name

Dim cLayViews As Layout2DViews
Set cLayViews = oLaySheet2.Views

Dim oLayView1, oLayView2 As Layout2DView
Set oLayView1 = cLayViews.ActiveView        'Retrieves the active view
Set oLayView2 = cLayViews.Item("Front view")'Retrieves a view using its name
```

The **Layout2DView** object aggregates the same objects than the **DrawingView** object, except the generative object: **DrawingGenView**.

Use the **GetItem** method of the **Layout2DRoot** object to retrieve the **Layout2DServices** object.

```
Dim oLayRoot As Layout2DRoot
...
Dim oLayServices As Layout2Services
Set oLayServices = Layout2DRoot.GetItem("CATLayout2DServices")
```

# Importing a Drawing into a New 2D Layout

This use case primarily focuses on the methodology to import a Drawing into a New 2D Layout.

---

Before you begin :

- You should first launch CATIA and import the `SkateDrawing.3dxml` file supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScd2dlFor3DDesign\samples\` where `InstallRootFolder` is the directory where the CAA CD-ROM is installed.

Where to find the macro : CAAScd2dlUcImpDrwTo2DLayoutSource.htm

This use case:

1. Gets the Drawing information
2. Creates the 3D Shape and its 2D Layout
3. Copies the Drawing Sheets in the 2D Layout

1. **Gets the Drawing information**

```
' Gets the drawing Editor
Dim drwEditor As Editor
Set drwEditor = CATIA.ActiveEditor

' Gets the drawing Root
Dim drwRoot As DrawingRoot
Set drwRoot = drwEditor.ActiveObject
```

```
' Gets the drawing Title
Dim drwRepRef As VPMRepReference
Set drwRepRef = drwRoot.Parent

Dim drwTitle As String
drwTitle = drwRepRef.GetAttributeValue("V_Name")
...
```

2. **Gets the Root Occurrence**

```
 ...
' Gets the PLMNewService
Dim oNewService As PLMNewService
Set oNewService = CATIA.GetSessionService("PLMNewService")

' Creates the 3D Shape representation
Dim shapeEditor As Editor
oNewService.PLMCreate "3DShape", shapeEditor

' Changes its Title
Dim shapeRepRef As VPMRepReference
Set shapeRepRef = shapeEditor.ActiveObject.Parent

shapeRepRef.SetAttributeValue "V_Name", drwTitle

' Creates the 2D Layout
Dim shapePart As Part
Set shapePart = shapeEditor.ActiveObject

Dim shapeLayRootFact As Layout2DFactory
Set shapeLayRootFact = shapePart.GetItem("CATLayoutRootFactory")

Dim shapeLayRoot As Layout2DRoot
Set shapeLayRoot = shapeLayRootFact.Create2DLayout(drwRoot.Standard)
...
```

The 3D Shape has the same Name as the Drawing and the 2D Layout has the same Standard as the Drawing.

3. **Copies the Drawing Sheets in the 2D Layout**

The UC copies each Sheet in the 2D Layout.

```
...
For i = 1 To drwRoot.Sheets.Count
  Dim drwSheet As DrawingSheet
  Set drwSheet = drwRoot.Sheets.Item(i)

  ' Creates a new Layout Sheet
  shapeLayRoot.Sheets.Add drwSheet.Name

  ' Gets the Sheet Views
  Dim tabToImport() As Variant
  ReDim tabToImport(drwSheet.Views.Count - 3)
  For j = 3 To drwSheet.Views.Count
     Set tabToImport(j - 3) = drwSheet.Views.Item(j)
  Next

  ' Gets the 2D Layout Services
  Dim layoutServices ' As Layout2DServices
  Set layoutServices = shapeLayRoot.GetItem("CATLayout2DServices")

  ' Imports the Views in the 2D Layout
  Dim tabImported() As Variant
  ReDim tabImported(drwSheet.Views.Count - 3)
  layoutServices.ImportFromDrawing tabToImport, CatImportAll, tabImported
Next
...
```

The method ImportFromDrawing copies the CATSafeArrayVariant tabToImport Drawing Views into the Active 2D Layout Sheet and copies the created 2D Layo tabImported.

Then, removes the First 2D Layout Sheet.

```
shapeLayRoot.Sheets.Remove 1
```

The first 2D Layout Sheet is created automaticaly with the 2D Layout and it is empty.

Fig.1: The Drawing

**Sheet 1** **Sheet 2**

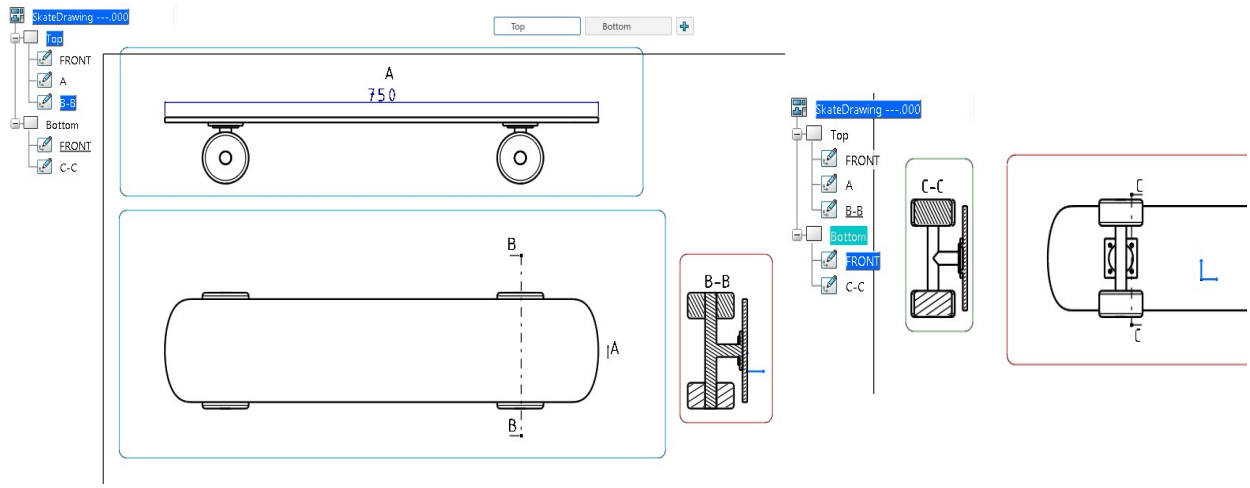



Fig.2: The generated 2D Layout

**Sheet 1** **Sheet 2**

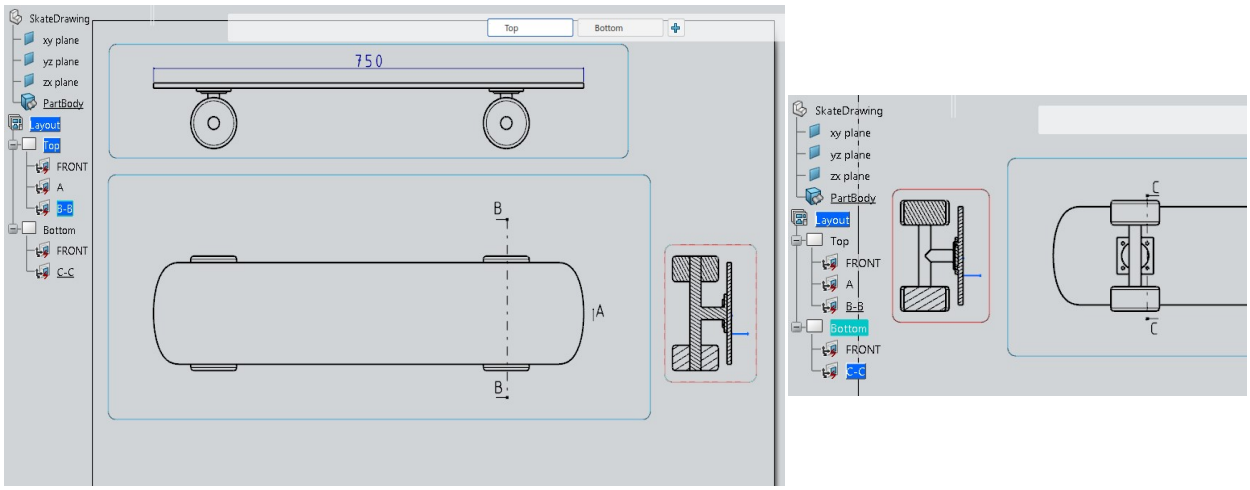
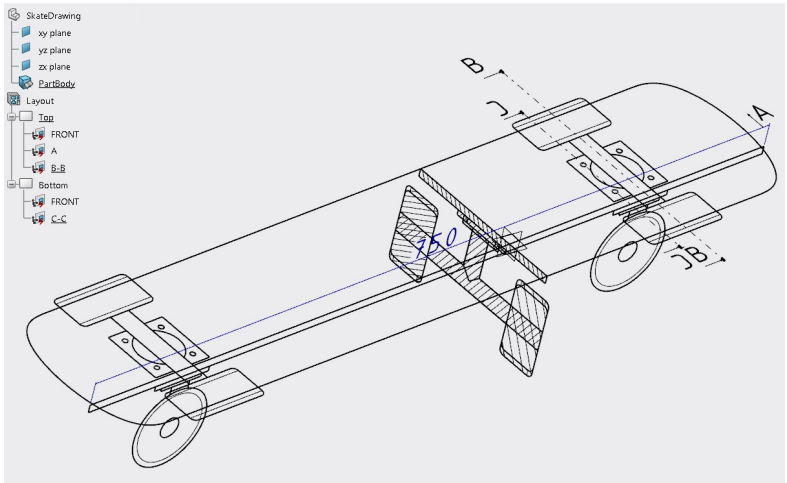
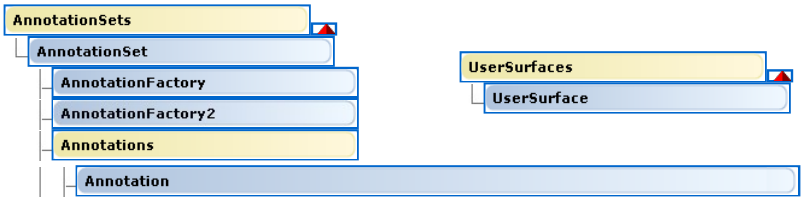


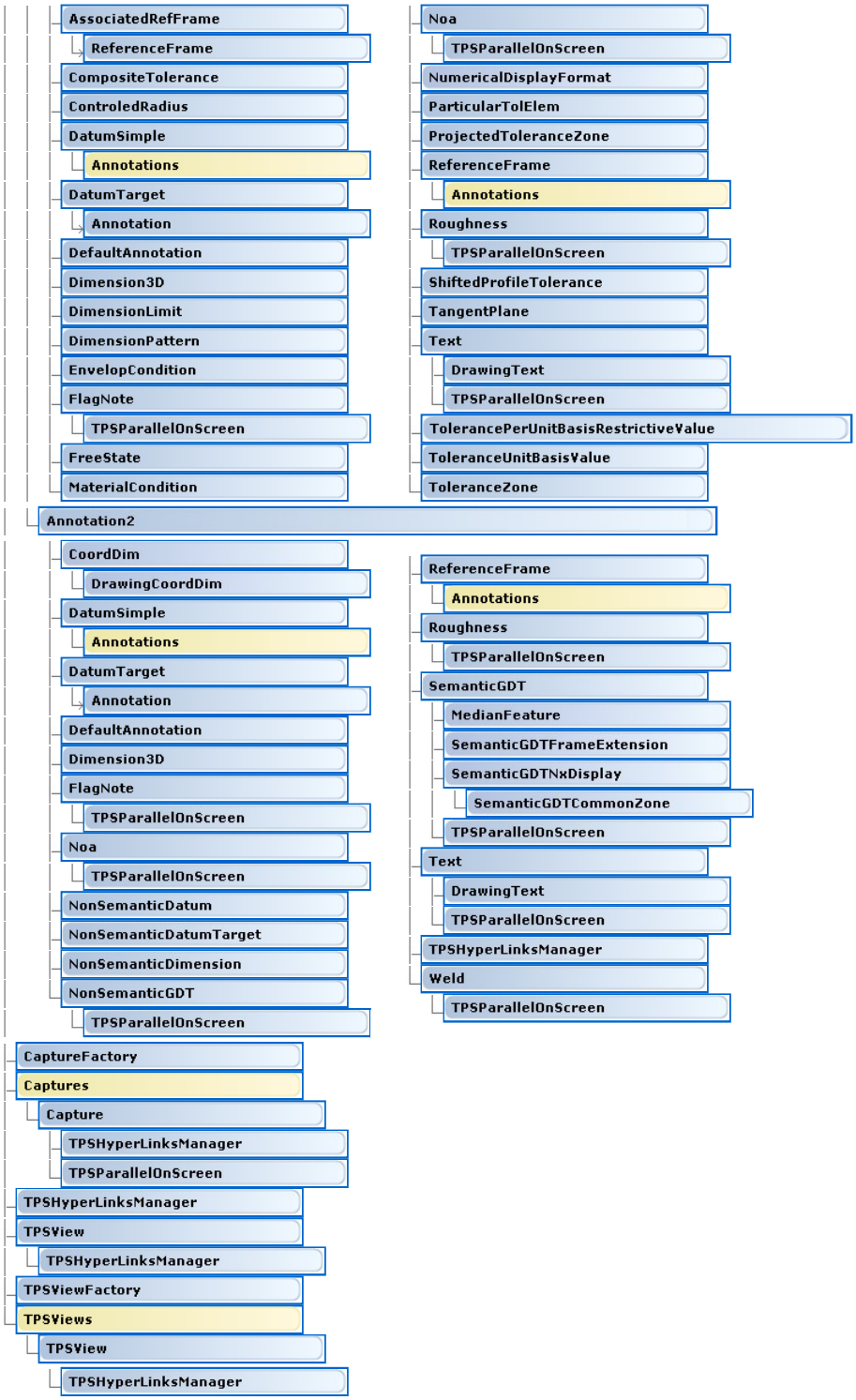

Fig.3: The generated Part




# 3D Tolerance & Annotation Object Model Map

See Also Legend






3D Tolerance & Annotations objects are obtained through two root objects:

- A collection of **AnnotationSet** objects.
- A collection of **UserSurface** objects.

Both can be obtained directly under the **Part** object of a 3D shape representation:

```
Set cAnnotationSets = oPart.AnnotationSets
Set cUserSurfaces = oPart.UserSurfaces
```

The **AnnotationSet** object aggregates an **Annotations** collection object and a **Captures** collection object.

A **Capture** object is a persistent viewpoint of a group of annotations.

An **Annotation** object always aggregates one of the objects that are listed on the left side of the diagram, that are: **Text, FlagNote, Roughness**, **Noa**(Note Object Attribute), **DatumSimple**, **DatumTarget**, and **ReferenceFrame** objects.

Depending on its type, the annotation may also aggregate one of the objects that are listed on the right side of the diagram. To access those aggregated objects, first check for availability using the corresponding `IsA/HasA` method and then get the object through the dedicated method. For example, to access a **ToleranceZone** object on an **Annotation** object, proceed as follows:

```
Dim oAnnotation As Annotation
Set oAnnotation= ...
If (oAnnotation.IsAToleranceZone()) Then
  Dim oToleranceZone As ToleranceZone
  Set oToleranceZone = oAnnotation.ToleranceZone()
End If
```

The **DatumTarget** object refers to a **DatumSimple** object through its **Datum** property.

# Creating a Ditto NOA & a Weld on the XY Plane

This use case primarily focuses on the methodology to create 2 non semantic features:

1. a Ditto NOA with the option "Stick Ditto perpendicular to geometry",
2. a Weld feature.

---

Before you begin :

- You should first launch CATIA and import the `CAAScdTpiUcDittoNoaWeldCreation.3dxml` file supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdTpiFTA\samples\` where `InstallRootFolder` is the directory where the CAA CD-ROM is installed.
- You will have to uncheck in your preferences the option "*Disable template creation*" in *"Preferences / Mechanical / 3D Tolerancing & Annotations"* within the *"Administration"* tab page to enable NOA creation.

Where to find the macro : CAAScdTpiUcDittoNoaWeldCreationSource.htm

This use case is illustrating:

1. Selects the Ditto from the 2DLayout
2. Creates the Ditto NOA entity
3. Provides a text content to be placed in the Ditto instance
4. Creates the Weld feature
5. Retrieves the 2D Annotation to adjust some characteristics

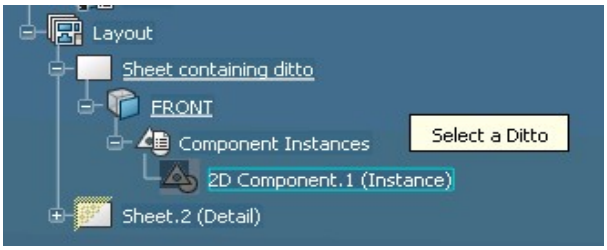
1. **Selects the Ditto from the 2DLayout**

   Lets the user choose the Ditto component to constitute display of the NOA entity

   ```
   ...
   ' Trigger the selection processing
   Dim Status, InputObjectType(0)

   InputObjectType(0) = "AnyObject"
   Status = Selection.SelectElement( InputObjectType, "Select a Ditto", False)

   If (Status = "Cancel") Then Exit Sub

   ' Retrieve the ditto
   Dim oSelectionItem As SelectedElement
   Set oSelectionItem = Selection.Item( 1 )

   Dim oDitto As DrawingComponent
   Set oDitto = oSelectionItem.Value

   ...
   ```

   The figure (Fig.1) is illustrating how the selection is taking place; notice that Ditto is directly picked from the Layout specification tree under the "*Component Instances*" node

   Fig.1: The selection of the Ditto Component

2. **Creates the Ditto NOA entity**

   Process to the NOA creation; this goes through the enumertaed steps:

   - Creates the Annotation Set based on the "ASME" Standard,
   - Accesses to the AnnotationFactory2 from the Annotation Set,
   - Creates a User Feature entity using a Reference generated from the XY Plane of the Part,
   - Invoke the method from the factory to create the NOA Ditto; final argument is set to TRUE to specify that "Stick Ditto perpendicular to geometry" option is required.

   ```
   ...
   ' Get the Part root feature
   Dim oPart as Part
   Set oPart = CATIA.ActiveEditor.ActiveObject

   ' Create a new ASME Annotation Set
   Dim oAnnotationSets As Collection
   Set oAnnotationSets = oPart.AnnotationSets
   Dim oAnnotationSet as AnnotationSet
   ```

```
Set oAnnotationSet = oAnnotationSets.Add("ASME")

' Retrieve the factory
Dim oFactory as AnnotationFactory2
Set oFactory = oAnnotationSet.AnnotationFactory2

' Create a reference on the XY Plane of the Part
Dim oReferenceOnXYPlane as Reference
Set oReferenceOnXYPlane = oPart.CreateReferenceFromObject( oPart.OriginElements.PlaneXY )

' Make the user surface corresponding to this reference
Dim oUserSurface As UserSurface
Set oUserSurface = oPart.UserSurfaces.Generate( oReferenceOnXYPlane )

' Create the NOA Ditto with option "Stick Ditto perpendicular to geometry"
Dim oNewDittoNOA as Annotation2
Set oNewDittoNOA = oFactory.CreateDittoNOA( oUserSurface, "VB_DittoNOA", oDitto , TRUE )
...
```

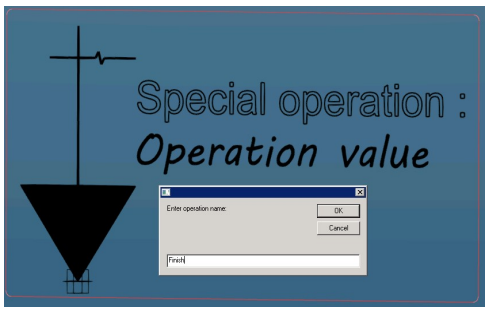
In case of wrong setting environment, macro execution is stopped by the error panel displayed by (Fig.2)

Fig.2: The Error raised consequent to wrong option activation.


3. **Provide a text content to be placed in the Ditto instance**

Drawing Ditto Annotation composing the NOA representation is accessible using the GetDitto method; next lines are demonstrating how InputBox is employed to
request text used to supersede Ditto modifiable text.

Fig.3: Input Box for text replacement in Ditto.




```
...
 ' Retrieve the ditto as component of this newly created NOA
Dim o2DComponentInNOA As DrawingComponent
Set o2DComponentInNOA = oNewDittoNOA.Noa.GetDitto( )

' Retrieve the modifiable text of the ditto
Dim oText As DrawingText
Set oText = o2DComponentInNOA.GetModifiableObject( 1 )

' Modify the modifiable text value
Dim ReturnValue As String
ReturnValue = InputBox( "Enter operation name: ", "", "New Value For Text")
oText.Text = ReturnValue
...
```

The Drawing Component object is employed in the rest of this script to modify the scale and the annotation orientation.

4. **Creates the Weld feature**

```
 ...
' Create a new Weld feature
Dim oNewWeld As Annotation2
Set oNewWeld = oFactory.CreateWeld( oUserSurface )
...
```

The CreateWeld method returns a nude Weld entity as Annotation2 object.

5. **Retrieves the 2D Annotation to adjust some characteristics**

```
...
```

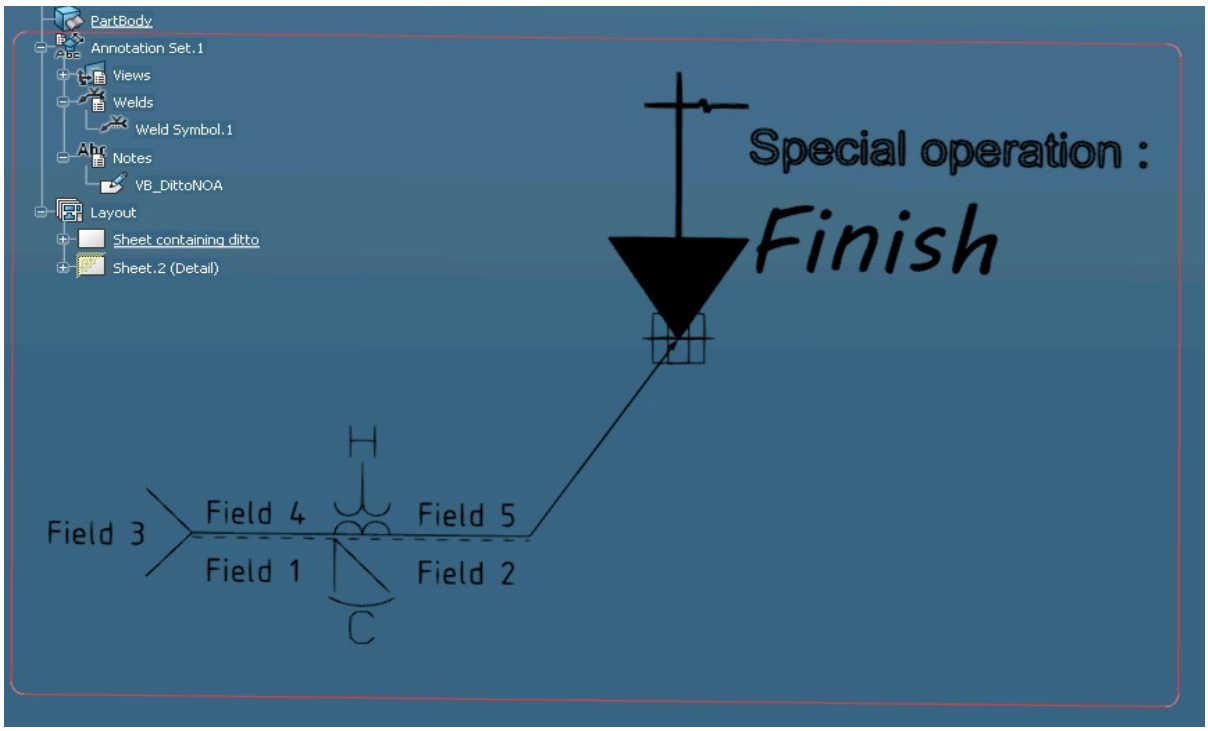
```
' Change Weld content
Dim oWeldingVisu As DrawingWelding
Set oWeldingVisu = oNewWeld.Weld.Get2DAnnot
...
```

Then, apply some changes to the Drawing Weld to modify its visualization.

```
...
oWeldingVisu.GetTextRange( catWeldingFieldOne ).Text = "Field 1"
oWeldingVisu.GetTextRange( catWeldingFieldTwo ).Text = "Field 2"
oWeldingVisu.GetTextRange( catWeldingFieldThree ).Text = "Field 3"
oWeldingVisu.GetTextRange( catWeldingFieldFour ).Text = "Field 4"
oWeldingVisu.GetTextRange( catWeldingFieldFive ).Text = "Field 5"
oWeldingVisu.GetTextRange( catWeldingFieldSix ).Text = "Field 6"
oWeldingVisu.GetTextRange( catWeldingFieldSeven ).Text = "Field 7"
oWeldingVisu.SetSymbol catHVGrooveWelding, catFirstWelding
oWeldingVisu.SetSymbol catRechargWelding, catSecondWelding
oWeldingVisu.SetAdditionalSymbol catConvexWelding, catFirstWelding
oWeldingVisu.SetAdditionalSymbol catSmoothWelding, catSecondWelding
oWeldingVisu.SetFinishSymbol catDftLetterCWelding, catFirstWelding
oWeldingVisu.SetFinishSymbol catDftLetterHWelding, catSecondWelding
oWeldingVisu.IdentificationLineSide = catWeldingDown
oWeldingVisu.WeldingSide = catWeldingDown
oWeldingVisu.WeldingTail = catDftWeldingTailYES

' Apply new position
oWeldingVisu.x = -70.
oWeldingVisu.y = -30.
...
```

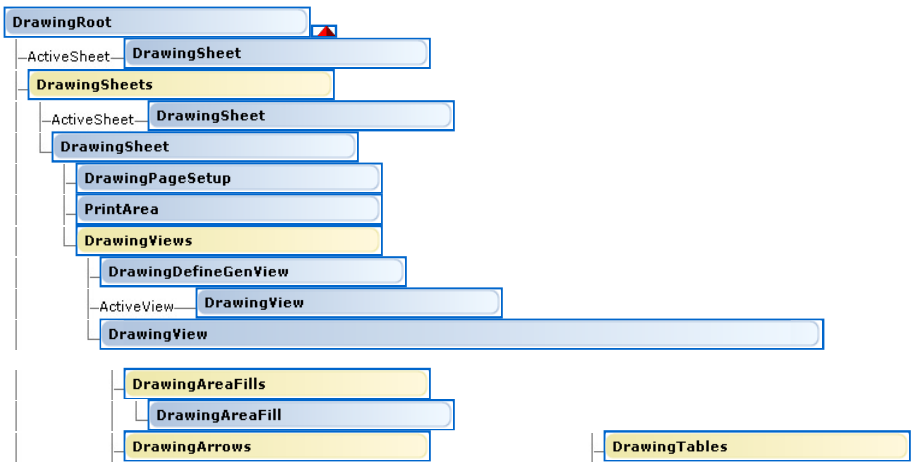
See the final result of this script execution (Fig.4).

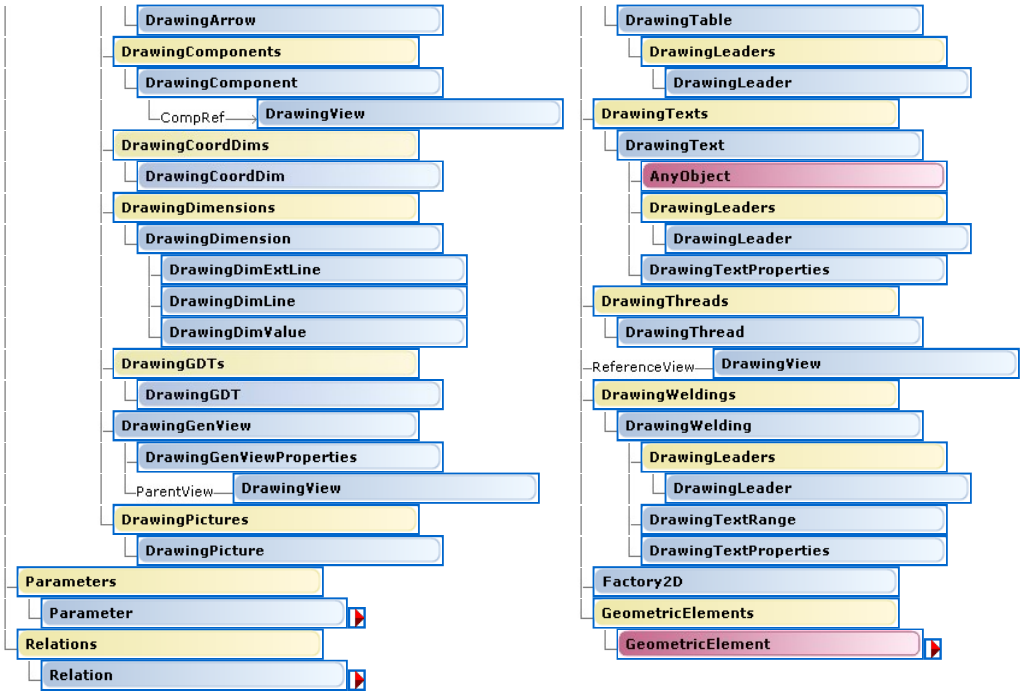
Fig.4: The Script expected outcome.




# Drafting Object Model Map

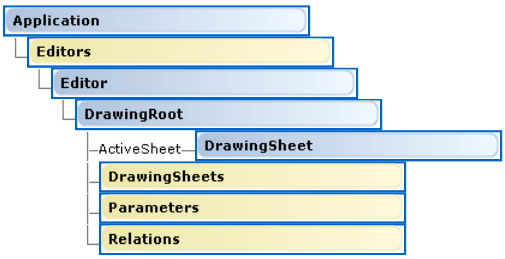
See Also Legend






# DrawingRoot Object

See Also Legend Use Cases Properties Methods




Represents the root object of a drawing representation.

The root object of a drawing representation can be retrieved thanks to the **ActiveObject** property of the drawing representation editor once the drawing representation is the active one.

```
Dim oDrwRoot As DrawingRoot
Set oDrwRoot = CATIA.ActiveEditor.ActiveObject
```

## Using the DrawingRoot Object

The following properties of the **DrawingRoot** object are described in this section:

- **ActiveSheet** property
- **Sheets** property.

### ActiveSheet Property

Use the **ActiveSheet** property to retrieve the active **Sheet** object:

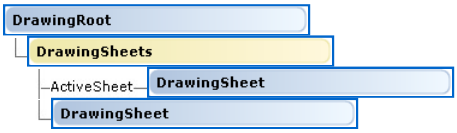
```
Dim oSht As Sheet
Set oSht = CATIA.ActiveEditor.ActiveObject.ActiveSheet
```

Use the **ActiveSheet** property along with the **Sheets** collection object to set the active sheet. The following example activates the second sheet in the collection.

```
Dim oSht As Sheet
Set oSht = CATIA.ActiveEditor.ActiveObject.Sheets.Item(2)
CATIA.ActiveEditor.ActiveObject.ActiveSheet = oSht
```

# DrawingSheets Collection Object

See Also Legend Use Cases Properties Methods




A collection of all the **DrawingSheet** objects that are currently open in a **DrawingRoot** object.

### Using the DrawingSheets Collection Object

Use the **Add** method to create a new drawing sheet and add it to the collection. The following example adds a new, empty drawing sheet to the active drawing representation.

```
Dim cDrawingSheets As DrawingSheets
cDrawingSheets.Add("MySheet")
```

Use the **AddDetail** method to add a detail drawing sheet and add it to the collection. The following example adds a new, empty drawing sheet to the active drawing representation.

```
Dim cDrawingSheets As DrawingSheets
...
cDrawingSheets.AddDetail("MyDetailSheet")
```

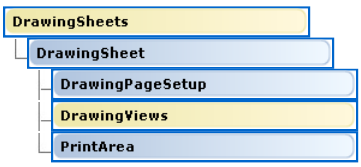
Use the **Item** method to retrieve a drawing sheet from the collection.

```
Dim cDrawingSheets As DrawingSheets
...
cDrawingSheets.Item(index)
```

Where index is the drawing sheet name or index number in the **DrawingSheets** collection object.

## DrawingSheet Object

See Also Legend Use Cases Properties Methods




Represents a drawing sheet object of a drawing representation.

All the **DrawingSheet** objects of the drawing representation are contained in the **DrawingSheets** collection object. The **DrawingSheet** object contains in turn a **DrawingViews** collection object that contains all the views of the sheet.

When a drawing sheet is created, it contains two drawing views:

1. The main view
2. The background view.

These two views are superimposed. They can be accessed as the first and second items of the **DrawingViews** collection object respectively. The main view coordinate system is used as the sheet coordinate system.

A drawing sheet has a page setup that defines the sheet attributes, such as its size, margins, orientation, and so forth. This page setup can be retrieved thanks to the **PageSetup** read only property.

### Using the DrawingSheet Object

Use the **Views** property to return the **DrawingViews** collection object.

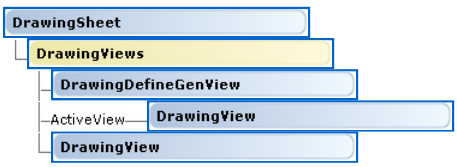
```
Dim cDrawingViews As DrawingViews
Set cDrawingViews = CATIA.ActiveEditor.ActiveObject.ActiveSheet.Views
```

Use the **PageSetup** property to retrieve the **DrawingPageSetup** object representing the sheet setup attributes.

```
Dim oSheetSetup As DrawingPageSetup
Set oSheetSetup = CATIA.ActiveEditor.ActiveObject.ActiveSheet.PageSetup
```

## DrawingViews Collection Object

See Also Legend Use Cases Properties Methods




A collection of all the **DrawingView** objects that are in a **DrawingSheet** object.

### Retrieving the DrawingViews Collection Object

Use the **Views** property to return the **DrawingViews** collection object.

```
Dim cDrawingViews As DrawingViews
Set cDrawingViews = CATIA.ActiveEditor.ActiveObject.ActiveSheet.Views
```

### Using the DrawingViews Collection Object

Use the **Add** method to create a new, empty drawing view and to add it to the collection. The following example adds a new, empty drawing view to the active drawing sheet. This drawing view is an interactive view.

```
Dim cDrawingViews As DrawingViews
...
cDrawingViews.Add("Front View")
```

Use the **Item** method to retrieve a drawing view from the collection.

```
Dim cDrawingViews As DrawingViews
...
cDrawingViews.Item(index)
```

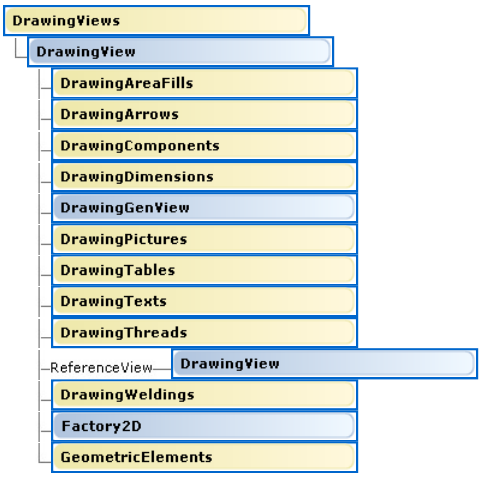
where index is the drawing view name or index number in the **DrawingViews** collection object.

Use the **DrawingDefineGenView** property to return the generative view factory.

```
Dim cDrawingViews As DrawingViews
...
Dim oGenViewFactory As DrawingDefineGenView
Set oGenViewFactory = cDrawingViews.DrawingDefineGenView
```

# DrawingView Object

See Also Legend Use Cases Properties Methods




Represents a drawing view of a drawing sheet in a drawing representation.

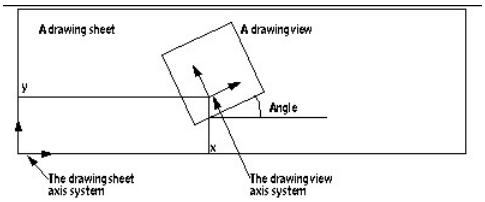
The **DrawingView** object accommodates two types of views:

1. The interactive view
2. The generative view.

The interactive view contains 2D CAD data that do not refer to any 3D data, while the generative view contains data created from 3D shape representations through assembly definitions. Any objects of an interactive view can be found in a generative view. On the opposite, an interactive view cannot aggregate the **DrawingGenView** object. The **IsGenerative** method returns whether the drawing view is a generative view.

## Interactive Views

The drawing view is placed in the drawing sheet using the following properties:




The **x**, **y**, and **Angle** properties are used to place the drawing view in the drawing sheet. The drawing sheet coordinate system is the one of its main view. The drawing view has also a **Scale** property.

Using the **Activate** method, you can make the drawing view the active one.

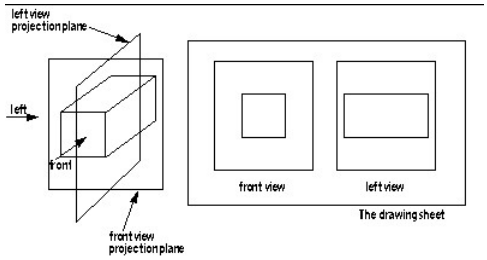
The following table summarizes the different interactive view types along with their ability to have a parent view, and if this parent view can also be a reference view.

| View Types | Parent View | Reference View |
|---|---|---|
| Front View | No | No |
| Isometric View | No | No |
| Projection View | Yes | Yes |

Let's now detail the different type of views.

A drawing view can be used as a parent view for other drawing views. When you create a drawing view, for example from a part, you first create a front view, and then you can create a left, right, top, or bottom drawing view from this front view. The left, right, top, or bottom view is called a projection view. The front view is used as a parent view to determine how the 3D shape representation is projected in the projection view. For example, left means projecting the part on a vertical plane which is perpendicular to the front view projection plane, and which is seen from the left.

A drawing view is strongly linked to its parent view. If the parent view is updated because the 3D shape representation it displays has changed, or if you change its scale, all the drawing views which have this view as parent view are changed accordingly.




The front view is also used as a reference by the left view for positioning. If you want to move the left view, it is constrained to move horizontally to remain a left view of the front view. The left view can access its reference view by means of the **ReferenceView** property.

The **AlignedWithReferenceView** and **UnAlignedWithReferenceView** methods enable you to align and deactivates the alignment with respect to the reference view.

## Generative Views

All generative views are created using the **DrawingDefineGenView** object.

The following table summarizes the different generative view types along with their ability to have a parent view, and if this parent view can also be a reference view. Each of them can be created thanks to a dedicated creation method of the **DrawingDefineGenView** object.

| View Types | Parent View | Reference View | Creation Methods |
|---|---|---|---|
| Front View | No | No | **DefineFrontView** |
| Isometric View | No | No | **DefineIsometricView** |
| Projection View | Yes | Yes | **DefineProjectionView** |
| Section View | Yes | Yes | **DefineSectionView** |
| Circular Detail View | Yes | No | **DefineCircularDetailView** |
| Polygonal Detail View | Yes | No | **DefinePolygonalDetailView** |
| Unfolded View | No | No | **DefineUnfoldedView** |
| Stand-alone View | No | No | **DefineStandAloneSection** |
| Auxiliary View | Yes | No | **DefineAuxiliaryView** |

For example, create a generative front view as follows.

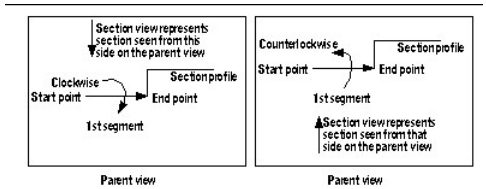
```
' Retrieve the DrawingDefineGenView factory object
Dim cDrawingViews As DrawingViews
Set cDrawingViews = CATIA.ActiveObject.ActiveSheet.Views
Dim oGenViewFactory As DrawingDefineGenView
Set oGenViewFactory = cDrawingViews.DrawingDefineGenView

' Define the generative view input data
ReDim ListOfinstances(10)   ' The list of product instances
...
ReDim ProjectionPlane(5)    ' The projection plane
...
Dim oDrawingGenViewProperties As DrawingGenViewProperties
...
' Create the generative view
Dim oGenView As DrawingView
Set oGenView = oGenViewFactory.DefineFrontView( _
     100, 100,                  ' The view anchor point coordinates
     ListOfinstances,           ' The list of product instances
     ProjectionPlane,           ' The projection plane
     "",                        ' The generative style
     True,                      ' Forces the view to be updated
     oDrawingGenViewProperties) ' The view properties
```

A front view or an isometric view are defined using:

- The coordinates of the view anchor point in the sheet coordinate system.
- A list of product instances that refer to the 3D shape representations to project.
- The projection plane thanks to two 3D vectors.
- A generative view style defined by the current standard.
- A flag to indicate whether the view should be updated after being created.
- A set of generative view properties held by a **DrawingGenViewProperties** object.

The other views are all defined by giving and additional information:

- A projection view is defined using a view type which can be left, right, top, bottom, or rear using the **CatProjViewType** enumeration.
- A section view or section cut is defined using a section profile in the parent view and passed as an array of point coordinates expressed with respect to the parent view axis system. Other information is required, namely the section type, that is whether the section is a section cut or a section view, the profile type, that is whether the section is aligned or offset, and the drawn side of the section. This drawn side is determined as follows: the parent view is rotated clockwise or counterclockwise around the first segment of the section profile oriented from its start point to its end point.



- A detail view is defined from a parent view and using a clipping circle or closed polygon.
- An unfolded view is dedicated to sheet metal parts.
- A stand-alone view is defined using a profile made up of 3D points defined using their coordinates.
- An auxiliary view is defined from a parent view and using a line which defines the trace of the auxiliary view projection plane in the parent view.

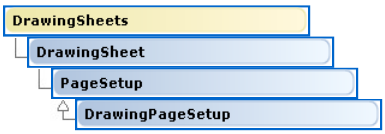
### Using the DrawingView Object

Use **x** and **y** properties to set or retrieve the x and y coordinates of the view coordinate system. The following example sets these coordinates to 260mm and 120mm respectively.

```
Dim oView As DrawingView
Set oView = CATIA.ActiveObject.ActiveSheet.ActiveView
oView.x = 260
oView.y = 120
```

## DrawingPageSetup Object

See Also Legend Use Cases Properties Methods




Represents the page setup of a drawing sheet of a drawing representation.

The page setup of a drawing sheet is the object that stores data which defines how the drawing sheet will be actually printed on paper. This data includes namely the paper size, the orientation, the bottom, top, right, and left margins, the zoom factor, the banner, the printing quality, the choice of the orientation, the ability to either select the appropriate printer format to fit the sheet format or to zoom the sheet in or out to fit the printer format.

The **DrawingPageSetup** object inherits most of its properties from the **PageSetup** object.

### Using the DrawingPageSetup Object

Use the **FitToPrinterFormat** property to zoom the active sheet in or out to fit the printer format.

```
CATIA.ActiveEditor.ActiveObject.ActiveSheet.DrawingPageSetup.FitToPrinterFormat = True
```

Use the **ChooseBestOrientation** property to activate the ability to choose the best orientation to fit the printer format.

```
CATIA.ActiveEditor.ActiveObject.ActiveSheet.DrawingPageSetup.ChooseBestOrientation = True
```

## Creating a Generative View from 3D Data

This use case primarily focuses on the methodology to create a Generative View of a Part Body in an Assembly Context.

---

Before you begin :

- You should first launch CATIA and import the Skate`board.3dxml` file supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdDriDrafting\samples\` where `InstallRootFolder` is the directory where the CAA CD-ROM is installed.

Where to find the macro : CAAScdDriUcGenViewOnPartBodySource.htm

This use case can be divided in seven steps:

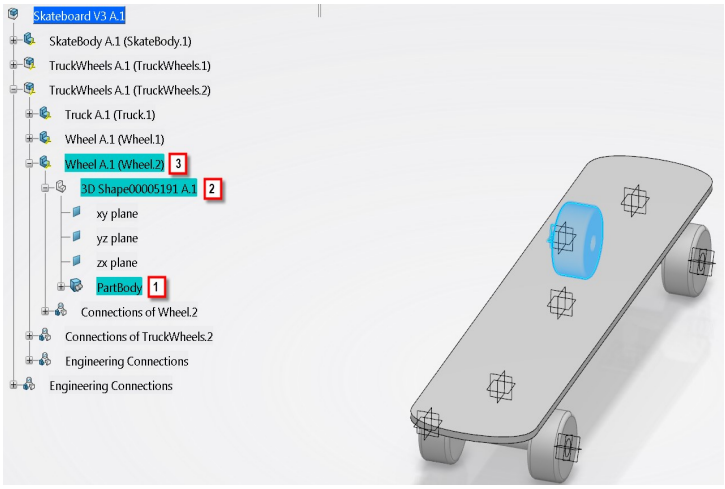
1. Selection of the required parts
2. Creation the Part Body Link
3. Definition the View Projection Plan
4. Creation the New Drawing Representation
5. Creation the Generative Front View
6. Modification the Generative View Link
7. Update of the Generative View

1. **Selection of the required parts**

   The body Part link creation requires, in this order :
   1 - The Part Body
   2 - The Product Representation Instance of the Reference on which the Part Body is defined
   3 - The first Sub-Product containing the RepInstance

   Before you start the macro, you have to select these three elements as shown in the figure 1.

   Fig.1: Selection of the parts required for the creation of the Part Body Link




2. **Creation the Part Body Link**

   First, the UC retrieves the selection and creates the Part Body Link by storring them in the right order in a CATSafeArrayVariant.

```
 ...
Dim mySel ' As Selection
Set mySel = CATIA.ActiveEditor.Selection

If (mySel.Count2 <> 0) Then
  Dim i As Integer
  Dim linkPartBody(2) As Variant

  For i = 1 To mySel.Count2

     If (TypeName(mySel.Item2(i).Value) = "VPMRepInstance") Then
        Set linkPartBody(1) = mySel.Item2(i).Value

     ElseIf (TypeName(mySel.Item2(i).Value) = "Body") Then
        Set linkPartBody(0) = mySel.Item2(i).Value

     ElseIf (TypeName(mySel.Item2(i).Value) = "VPMOccurrence") Then
        Set linkPartBody(2) = mySel.Item2(i).Value

     End If
  Next
  ...
```

   Then, the UC retrieves the drawing service object from the CATIA service.

```
 ...
  Dim myDrwServ ' As DrawingGenService
  Set myDrwServ = CATIA.GetSessionService("CATDrawingGenService")
...
```

   At last, it checks the Link integrity.

```
 ...
  If myDrwServ.CheckViewLinkIntegrity(linkPartBody) = False Then
     MsgBox "Select :" & vbCr & "- A Part Body" & vbCr & "- The Product Representation Instance of the Reference on which the Part
     Exit Sub
  End If
 ...
```

3. **Definition the View Projection Plan**

   In this step UC asks to select a Projection Plan.

```
...
Dim InputObjectType(0), Status
Dim projPlane ' As PlanarFace

InputObjectType(0) = "PlanarFace"
mySel.Clear
Status = mySel.SelectElement2(InputObjectType, "Select a projection plane", True)
If Status = "Cancel" Or Status = "Undo" Then
  Exit Sub
End If
Set projPlane = mySel.Item(1).Value
mySel.Clear
...
```

   Then, it stores the projection plane as a CATSafeArrayVariant with the coordinates of 2 vectors.

```
...
Dim firstAxis(2), secondAxis(2)
projPlane.GetFirstAxis (firstAxis)
projPlane.GetSecondAxis (secondAxis)

Dim myListOfPlane(5)
myListOfPlane(0) = firstAxis(0)
myListOfPlane(1) = firstAxis(1)
myListOfPlane(2) = firstAxis(2)
```

```
myListOfPlane(3) = secondAxis(0)
myListOfPlane(4) = secondAxis(1)
myListOfPlane(5) = secondAxis(2)
...
```

4. **Creation the New Drawing Representation**

   In this step UC creates the new Drawing.

   First, it creates a new Drawing Editor.

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

Dim oEditor As Editor
oNewService.PLMCreate "Drawing", oEditor
...
```

   Then, it gets the drawing root from the Editor.

```
...
Dim oDrawingRoot As DrawingRoot
Set oDrawingRoot = oEditor.ActiveObject
...
```

   It sets the drawing standard.

```
...
oDrawingRoot.Standard = "ISO"
...
```

   It sets the sheet style to the automatically created sheet.

```
...
oDrawingRoot.ActiveSheet.SheetStyle = "A0 ISO"
...
```

5. **Creation the Generative Front View**

   First, the UC retrieves the Generative View factory.

```
...
   Dim cDrawingViews As DrawingViews
   Set cDrawingViews = oDrawingRoot.Sheets.ActiveSheet.Views

   Dim oGenViewFactory As DrawingDefineGenView
   Set oGenViewFactory = cDrawingViews.DrawingDefineGenView
...
```

   Then, it defines the new created view as a Generative View.

```
...
   Dim myDefGenView ' As DrawingDefineGenView
   Set myDefGenView = cDrawingViews.DrawingDefineGenView
...
```

   It retrieves a Generative View Properties.

```
...
   Dim myGenViewProp As DrawingGenViewProperties
   Set myGenViewProp = myDrwServ.DrawingGenViewProp
...
```

   At last, it creates the Generative View of the Sub-Product.

   The Generative view can't be directly created on the Part Body. The solution here is to create the view on the sub product and then to modify its link.

```
...
   Dim myListOfPLMInst(0) As Variant
   Set myListOfPLMInst(0) = linkPartBody(2)

   Dim myNewFrontView As DrawingView
   Set myNewFrontView = myDefGenView.DefineFrontView(100, 100, myListOfPLMInst, myListOfPlane, "", False, myGenViewProp)
...
```

   The sixth argument is set to false to postpone updating the drawing after the link modification.

6. **Modification the Generative View Link**

   In this step UC modifies the link of the view created in the previous section.

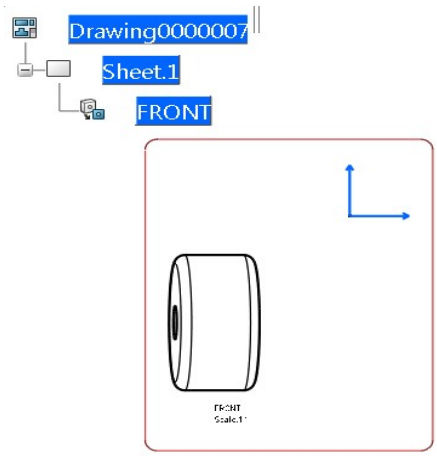
```
...
myNewFrontView.DrawingGenView.PutLinks 1, linkPartBody
...
```

7. **Update of the Generative View**

   At last, it updates the view.

```
...
myNewFrontView.DrawingGenView.Update
...
```

Fig.2: The Generated Drawing




# Generating Attribute Links Extracted from 3D Part Pointed by a Generative View in a Drawing Table

This use case primarily focuses on the methodology to Generate Attribute Links Extracted from 3D Part Pointed by a Generative View in a Drawing Table

---

Before you begin :

- You should first launch CATIA and import the `Skateboard.3dxml` file supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdDriDrafting\samples\` where `InstallRootFolder` is the directory where the CAA CD-ROM is installed.
- Open the Drawing in the Assembly and select the Generative View of the Wheel.
- Launch the following macro :

Where to find the macro : CAAScdDriUcGenAttrLinkFromGenViewSource.htm

This use case:

1. Retrieves the Product Reference pointed by the Generative Drawing View
2. Retrieves the Product Parameters
3. Creates and fills the Drawing Table

1. **Retrieves the Product Reference pointed by the Generative Drawing View**

   Before starting the macro, the Generative View in wich you want to create the Drawing Table must be selected.

   First, the UC Checks whether the View is Generative.

```
Dim myDrawingRoot As DrawingRoot
Set myDrawingRoot = CATIA.ActiveEditor.ActiveObject

If (myDrawingRoot.ActiveSheet.Views.ActiveView.IsGenerative) Then
...
```

   Then, determines on what the Generative View Link is.

```
...
Dim myGenView ' As DrawingGenView
Set myGenView = myDrawingRoot.ActiveSheet.Views.ActiveView.DrawingGenView

Dim nbInfoOnLink As Long
nbInfoOnLink = myGenView.GetNumberOfInfoForLink(1)
...
```

   A Link on an Occurence has one information and a Link on a PartBody has three.

   Gets the Generative View Link.

```
...
ReDim oInfoOnViewLink(nbInfoOnLink - 1) As Variant
myGenView.GetLink 1, oInfoOnViewLink
...
```

   Gets the Product Reference.

```
...
If (nbInfoOnLink = 1) Then
  Dim oProdOcc As VPMOccurrence
  Dim oProdRef As VPMReference

  Set oProdOcc = oInfoOnViewLink(0)
```

```
  Set oProdRef = oProdOcc.InstanceOccurrenceOf.ReferenceInstanceOf

ElseIf (nbInfoOnLink = 3) Then

  Dim oProdRepInst As VPMRepInstance
  Set oProdRepInst = oInfoOnViewLink(1)
  Set oProdRef = oProdRepInst.ReferenceInstanceOf.Father
...
```

If the Case of an Occurence, the method GetLink returns the Occurence. The Link of a Part Body returns in this order : the PartBody, the Product Representation Instance of the Reference on which the Part Body is defined and the first Sub-Product containing the RepInstance. The UC uses the Product Representation Instance to get the Product Reference.

2. **Retrieves the Product Parameters**

   To get the Parameters, the Part Body must be open in the session.

   First, the UC opens the Part Body.

```
 ...
' Gets the SearchService object
Dim oSearchService As SearchService
Set oSearchService = CATIA.GetSessionService("Search")

' Creates a query using the DatabaseSearch object
Dim oDBSearch As DatabaseSearch
Set oDBSearch = oSearchService.DatabaseSearch

' Gets the Product Reference PLM_ExternalID
Dim oProdPLMID As String
oProdPLMID = oProdRef.GetAttributeValue("PLM_ExternalID")

' Sets the type of object to query, a criterion, and trigger the search
oDBSearch.BaseType = "VPMReference"
oDBSearch.AddEasyCriteria "PLM_ExternalID", oProdPLMID

' Launches the search
oSearchService.Search

' Retrieves the listed entities
Dim cPLMEntities As PLMEntities
Set cPLMEntities = oDBSearch.Results

Dim oPLMOpenService ' As PLMOpenService

' Gets the PLMOpen service & Open the first retrieved entity
Set oPLMOpenService = CATIA.GetSessionService("PLMOpenService")
Dim oEditor As Editor
oPLMOpenService.PLMOpen cPLMEntities.Item(1), oEditor
...
```

   The code above is usual code to search a Product in the Database and open it.

   Gets the Part Body Parameters from the activated Editor.

```
...
Dim oPart As Part
Set oPart = oEditor.ActiveObject
Dim myParameters As Parameters
Set myParameters = oPart.Parameters
...
```

   Closes the Part Body Window and returns to the Drawing.

```
...
Dim oWindow As Window
Set oWindow = CATIA.ActiveWindow
oWindow.Close
...
```

3. **Creates and fills the Drawing Table**

   First, the UC creates the Drawing Table.

```
...
Dim myView As DrawingView
Set myView = myDrawingRoot.ActiveSheet.Views.ActiveView

Dim myTable As DrawingTable
Set myTable = myView.Tables.Add(100, 100, 1, 2, 20, 150)

myTable.ComputeMode = CatTableComputeOFF
...
```

   The last line turns off the display of modifications in the Dawing Table.

   Fills the Drawing Table with the Parameters names.

```
...
```

```
For i = myParameters.Count To 1 Step -1
 ' Cuts the Parameter Name
 cutParameterName = Split(myParameters.Item(i).Name, "\")
 ' Filters the Parameters
 If (cutParameterName(UBound(cutParameterName)) <> "Activity" And cutParameterName(UBound(cutParameterName)) <> "Mode") Then
    ' Adds a row and columns if necessary
    myTable.AddRow 1

    If (nbColumns - 1 < (UBound(cutParameterName) - 2)) Then
       For k = nbColumns - 1 To (UBound(cutParameterName) - 2)
          myTable.AddColumn nbColumns
          nbColumns = nbColumns + 1
       Next
    End If

    ' Fills the Table with the Parameter Name and changes the Cell Layout
    For j = 2 To UBound(cutParameterName)
       myTable.SetCellString 2, (j - 1), cutParameterName(j)
       ' Changes the Cell Layout
       myTable.SetCellAlignment 2, (j - 1), CatTableMiddleCenter
       Set myText = myTable.GetCellObject(2, j - 1)
       myText.SetFontSize 0, 0, 10
    Next
...
```

The Drawing Table is filled from the bottom to the top : the method AddRow adds a row above the selected row.

Parameters Names contains their whole Path. Splitting it is necessary to choose the informations to display.

Creates the Attribute Link between the Drawing Table and the Part Body Parameters.

```
...
myTable.SetCellString 2, nbColumns, ""
Set myText = myTable.GetCellObject(2, nbColumns)
myText.InsertVariable 0, 0, myParameters.Item(i)
' Changes the Cell Layout
myTable.SetCellAlignment 2, nbColumns, CatTableMiddleCenter
myText.SetFontSize 0, 0, 10
...
```

The Drawing Text method "InsertVariable" creates a Link on a Parameter. To use it, you first need to create the Drawing Text with the method SetCellString and then to get the Drawing Text from the Cell with the method GetCellObject.

Iniatializes the Drawing Table Title and refresh the Drawing Table.

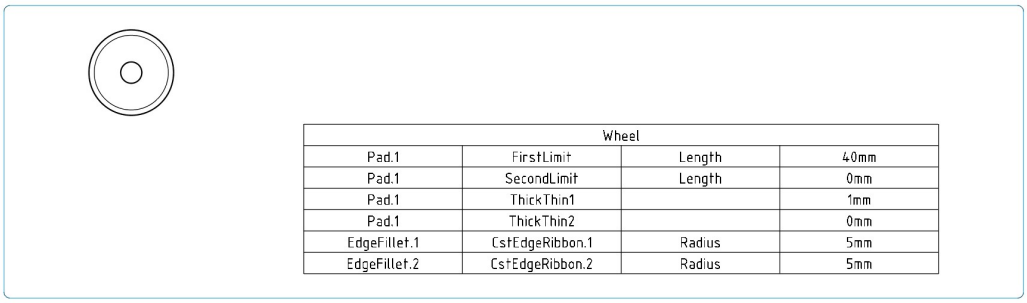
```
...
myTable.SetCellString 1, 1, ""
Set myText = myTable.GetCellObject(1, 1)
myText.InsertAttributeLink 0, 0, oProdRef, "VPMReference", "V_Name"
' Changes the Cell Layout
myText.SetFontSize 0, 0, 10
myTable.MergeCells 1, 1, 1, nbColumns
myTable.SetCellAlignment 1, 1, CatTableMiddleCenter
' Enables display of the Drawing Table modifications.
myTable.ComputeMode = CatTableComputeON
...
```

The Drawing Text method "InsertAttributeLink" creates a Link on PLM Attribute. The method to get the Drawing Text is the same as above.

## In Short :

**This use case shows how to create text attribute by using InsertVariable method for Knowledge Parmaterers or InsertAttributLink for PLM Attributes.**

Fig.1 : The Generated Drawing Table



| Wheel | | | |
|---|---|---|---|
| Pad.1 | FirstLimit | Length | 40mm |
| Pad.1 | SecondLimit | Length | 0mm |
| Pad.1 | ThickThin1 | | 1mm |
| Pad.1 | ThickThin2 | | 0mm |
| EdgeFillet.1 | CstEdgeRibbon.1 | Radius | 5mm |
| EdgeFillet.2 | CstEdgeRibbon.2 | Radius | 5mm |

# Creating an AreaFill Covering a Drawing View

This use case primarily focuses on the methodology to create an AreaFill Covering a Drawing View .

---

Before you begin :

- You should first launch CATIA and import the `SkateDrawing.3dxml` file supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdDriDrafting\samples\` where `InstallRootFolder` is the directory where the CAA CD-ROM is installed.

Where to find the macro : CAAScdDriUcAreaFillCoverViewSource.htm

This use case:

1. Selects the Drawing View to cover
2. Gets the Drawing View Position
3. Creates the AreaFill covering the Drawing View

1. **Selects the Drawing View to cover**

   Lets the user choose the Drawing View to cover

```
...
Dim drwView ' As DrawingView

Dim mySel ' As Selection
Set mySel = CATIA.ActiveEditor.Selection

mySel.Clear

Dim InputObjectType(0), Status
InputObjectType(0) = "DrawingView"

Status = mySel.SelectElement2(InputObjectType, "Select a view", True)
If Status = "Cancel" Or Status = "Undo" Then
Exit Sub
End If

Set drwView = mySel.Item(1).Value

mySel.Clear
...
```

   The method SelectElement2 allows the user to select a certain type of Element.

2. **Gets the Drawing View Position**

```
 ...
Dim viewSize(3) As Variant
drwView.Size viewSize
...
```

   The Drawing View method returns the Xmin/Xmax and Ymin/Ymax of the Drawing View Bounding Box.

3. **Creates the AreaFill covering the Drawing View**

   Sets the number of points of the AreaFill Contour.

```
...
Dim nbPoints(0) As Variant
nbPoints(0) = 4
...
```

   Then, sets the points of the AreaFill Contour.

```
...
Dim areafillContour(7) As Variant
Dim delta As Double
delta = 20
areafillContour(0) = viewSize(0) - delta
areafillContour(1) = viewSize(3) + delta
areafillContour(2) = viewSize(1) + delta
areafillContour(3) = viewSize(3) + delta
areafillContour(4) = viewSize(1) + delta
areafillContour(5) = viewSize(2) - delta
areafillContour(6) = viewSize(0) - delta
areafillContour(7) = viewSize(2) - delta
...
```

   The AreaFill is bigger than the Drawing View Bounding Box to fully cover it.

   Creates the AreaFill and changes its Pattern.

```
...
Dim myAreaFills ' As DrawingAreaFills
Set myAreaFills = drwSheet.Views.Item(1).AreaFills

Dim myAreaFill As DrawingAreaFill
Set myAreaFill = myAreaFills.Add(nbPoints, areafillContour)

myAreaFill.SetPattern "Yellow"
```

   See the different Patterns in the AreaFill Properties (Fig.2).

Fig.1: The Generated AreaFill

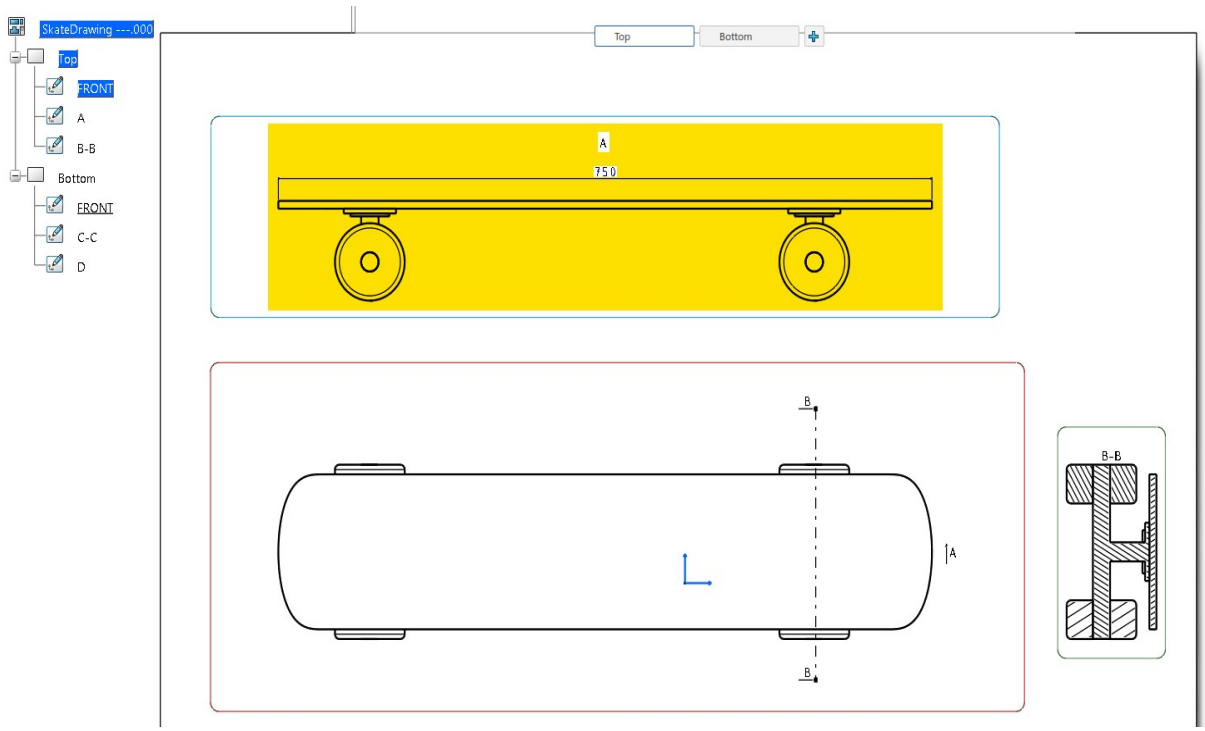
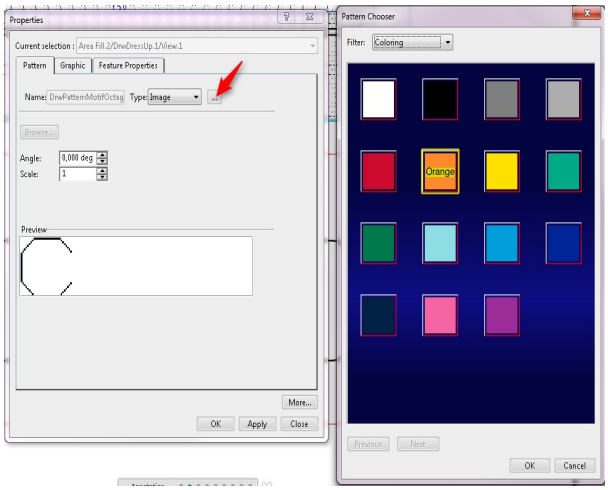
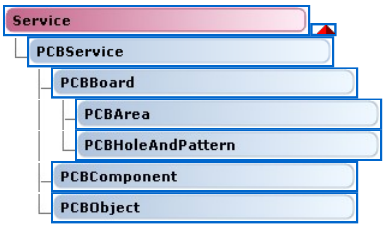



Fig.2: Patterns in the AreaFill Properties




# Circuit Board Design Object Model Map

See Also Legend




Circuit Board Design objects can be managed using the **PCBService** object. This object allows you to:

- Create a **PCBBoard** object or a **PCBComponent** object.
- Retrieve a **PCBObject** object.

Use the **CreateBoard** method to create a **PCBBoard** object in a 3D **Part** object of a 3D shape representation:

```
Dim oPCBService As PCBService
Set oPCBService = CATIA.ActiveEditor.GetService("PCBService")

Dim oPart As Part
Set oPart = CATIA.ActiveEditor.ActiveObject
```

```
Dim oPCBBoard As PCBBoard
Set oPCBBoard = oPCBService.CreateBoard(oPart)
```

# Flexible Circuit Board Design Object Model Map

See Also Legend


Flexible Circuit Board Design objects can be managed using the **FCBService** object. This object allows you to:

- Create a **FlexibleBoard** object.
- Retrieve a **FlexibleBoard** object.

Use the **CreateFlexibleBoard** method to create a **FlexibleBoard** object with an Axis System in a 3D **Part** object of a 3D shape representation:

```
Dim oFCBService As FCBService
Set oFCBService = CATIA.ActiveEditor.GetService("FCBService")

Dim oPart As Part
Set oPart = CATIA.ActiveEditor.ActiveObject

Dim oFlexiBoard As FlexibleBoard
Set oFlexiBoard = oFCBService.CreateFlexibleBoard(oPart, True)
```

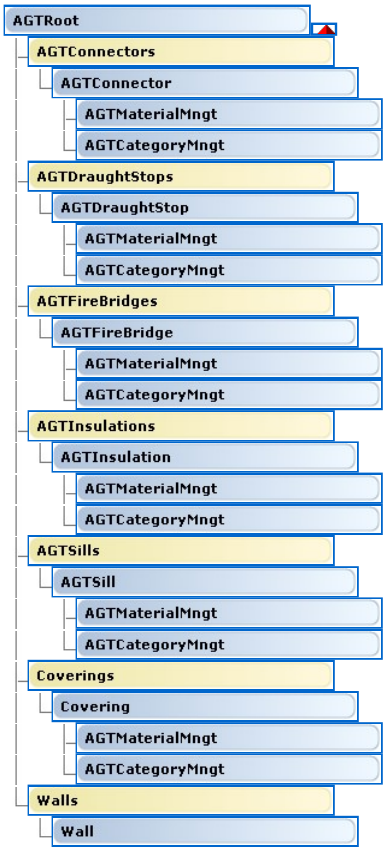
When set to **True**, the Boolean defined as the second argument of the **CreateFlexibleBoard** method allows you to create the Axis System.

Use the **create_FlexibleAreaArea** method of the **FlexibleBoard** object to create a constraint area as a **FlexibleArea** object.

```
Dim oHybridBody As HybridBody
Set oHybridBody = oPart.HybridBodies.Item("Constraint Area Profiles")
Dim oSketch As Sketch
Set oSketch = oHybridBody.HybridSketches.Item("Sketch.17")

Dim oBody As Body
Set oBody = oPart.Bodies.Item("Central")

Dim oArea As FlexibleArea
Set oArea = oFlexiBoard.create_FlexibleAreaArea(oBody, "MfDefault3DView", oSketch, "ROUTE_OUTLINE", "All", "Ident", "MCAD", 10, 5)
```

# Accommodation Design Object Model Map

See Also Legend




The **AGTRoot** object is the root object for Accommodation Design applications. It is retrieved from the **Part** object thanks to the **GetItem** method.

```
...
Dim oAGTRoot As AGTRoot
Set oAGTRoot = oPart.GetItem("CATAGTRoot")
...
```

The **AGTRoot** object aggregates an **AGTInsulation** collection object that contains **AGTInsulation** objects.

```
...
Dim cInsulations As AGTIinsulations
Set cInsulations = AGTRoot.Insulations

Dim oInsulationByName As AGTInsulation
Set oInsulationByName = cInsulations.Item("Insulation.1") 'Retrieves a Insulation using its name

Dim oInsulationByIndex As AGTInsulation
Set oInsulationByIndex = cInsulations.Item(1) 'Retrieves a Insulation using its index
...
```

# Accessing the Category and Material on Accommodation Objects

This use case primarily focuses on the methodology to access to the category and material on insulation object.

---

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdAgtUcInsulationModel.3dxml`files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdAgtAccommodation\samples\` where `InstallRootFolder` is the directory where the CAA CD-ROM is installed.

Related Topics
*<topic1>*
*<topic2>*

Where to find the macro: CAAScdAgtUcAccessCategoryMaterialSource.htm

This use case can be divided in eight steps:

1. Searches and opens model which is named as "InsulationModel"
2. Retrieves Part object
3. Retrieves collection of Insulation from part object
4. Retrieves a Insulation object from collection of insulation
5. Retrieves AGTMaterialMngt object
6. Get the material & Set the material
7. Retrieves AGTCategoryMngt object
8. Get the category & Set the category

1. **Searches and opens model which is named as "InsulationModel"**

   As a first step, the UC retrieves a model "Insulation Model" from DB and loads it and returns object of the Editor.

   ```
   ...
   Dim AGTPrdEditor  As  Editor
   OpenProduct AGTPrdEditor
   ...
   ```

   The function **`OpenProduct`** returns **`AGTPrdEditor`**, a Editor object. After searching and opening of Accommodation model from underlying database the current active editor is returned in **`AGTPrdEditor`**

2. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

   ```
   ...
   Set ObjPart = AGTPrdEditor.ActiveObject
   ...
   ```

3. **Retrieves collection of Insulation from part object**

   In this step UC retrieves colloction of insulation from part object.
   First, retrieves accommodation root object from part.
   Second, retrieves collocetion of insulation from root object.

   ```
   ...
   Dim MyRoot As AGTRoot
   Set MyRoot = ObjPart.GetItem("CATAGTRoot")
   Dim InsulationCollection As AGTInsulations
   Set InsulationCollection = MyRoot.Insulations
   ...
   ```

4. **Retrieves a Insulation object from collection of insulation**

   In this step UC retrieves Insulation object from collection of insulation.

   ```
   ...
   Dim ObjInsulationByIndex As AGTInsulation
   Dim ObjInsulationByName As AGTInsulation
   Set ObjInsulationByIndex = InsulationCollection.Item(1)              'Index of Insulation object
   Set ObjInsulationByName = InsulationCollection.Item("Insulation.2")  'Name of Insulation object
   ...
   ```

   There are two ways. First is Index, and the other is Name.
   This index is the rank of the insulation in the collection. The index of the first insulation is 1. The name is name of insulation object.

5. **Retrieves AGTMaterialMngt object**

   Get a AGTMaterialMngt object for the first insulation object.

   ```
   ...
   ```

```
Dim ObjMaterialMngt As AGTMaterialMngt
Set ObjMaterialMngt = ObjInsulationByIndex.AGTMaterialMngt
...
```

6. **Get the material & Set the material**

   Get and Set the material of the first insulation object.

```
...
AGTMaterialName = ObjMaterialMngt.GetMaterial
ObjMaterialMngt.SetMaterial "A-800"         'Material name of insulation
...
```

7. **Retrieves AGTCategoryMngt object**

   Get a AGTCategoryMngt object for the second insulation object.

```
...
Dim ObjCategoryMngt As AGTCategoryMngt
Set ObjCategoryMngt = ObjInsulationByName.AGTCategoryMngt
...
```

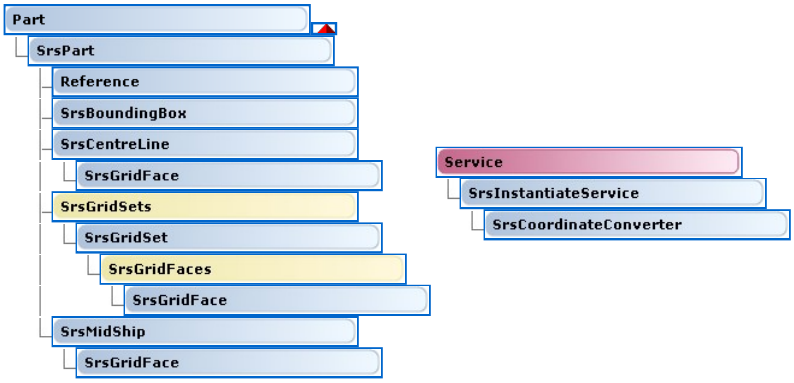
8. **Get the category & Set the category**

   Get and Set the material of the second insulation object.

```
...
AGTCategoryName = ObjMaterialMngt.GetCategory
ObjCategoryMngt.SetCategory "DeckPanelInsulation"      'Category name of insulation
...
```

# Space Referential Object Model Map

See Also Legend




The **SrsPart** object is retrieved from the **Selection** object containing a **Part** object using the **FindObject** method.

```
  ...
  Dim oSelection As Selection
  Set oSelection = CATIA.ActiveEditor.Selection

  Dim oSrsPart As SrsPart
  Set oSrsPart = oSelection.FindObject("CATIASrsPart")
...
```

The **SrsInstantiateService** object is retrieved from the **Editor** object. Then the **SrsCoordinateConverter** object is retrieved form the **SrsInstantiateService** object using the **CreateSrsCoordinateConverter** method.

```
...
  Dim oSrsService As SrsInstantiateService
  Set oSrsService = CATIA.ActiveEditor.GetService("SrsInstantiateService")

  Dim oSrsCoordConverter As SrsCoordinateConverter
  oSrsService.CreateSrsCoordinateConverter oSrsCoordConverter
...
```

# Retrieving SRS Data

This use case primarily focuses on the methodology to retrieve and edit SRS data.

---

Before you begin: Note that:

- Launch CATIA
- Assign "Project Dictionary" resource in "Common Geometry Resource" table in data setup. A sample resource `SpaceReferential_ShipBuilding_Dictionary.3dxml` is supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSrsSpaceReferenceSystem\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.
- Load a SRS part from database or create one before launching the macro. A sample `CAAScdSrsUcPart.3dxml` is supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSrsSpaceReferenceSystem\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.
- Go to "Space Reference System" application and keep product editor active before launching the macro.

Where to find the macro: CAAScdSpaceReferenceSystemSource.htm

Related Topics
Space Reference System Object Model Map
Launching an Automation Use Case

This use case can be divided in fourteen steps:

1. Retrieve the active editor of the Srs product
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves a grid set
5. Retrieves the group faces of the grid set
6. Retrieves a grid face in the grid set
7. Retrieves short name of the grid face
8. Sets short name of the grid face
9. Retrieves the category of the grid face
10. Sets a new category on the grid face
11. Retrieves the Reference hull surface of the SRS part
12. Retrieves the MidShip plane of the SRS part
13. Retrieves the CentreLine plane of the SRS part
14. Updates the Part object

1. **Retrieves the active editor of the Srs product**

   As a first step, open a SRS part in editor. The supplied sample "CAAScdSrsUcPart.3dxml" can be imported and open the "SRS_VB_TEST" 3DPart. Retrieve the active editor.

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

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SRSProdSel variable. To retrieve the Selection object **SRSPrdEditor** is used.

```
 ...
  Set SRSProdSel = SRSPrdEditor.Selection
 ...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set ObjSrsPart = SRSProdSel.FindObject("CATIASrsPart")
...
```

4. **Retrieves a SRS grid set object from the list of grid sets present in the SRS part**

   In this step UC retrieves the list of grid sets object ObjSrsGridSets variable.

```
...
  Set ObjSrsGridSets = ObjSrsPart.SrsGridSets
...
```

   From the the list of grid sets, individual grid set object ObjSrsGridSet is retrieved.

```
...
  Set ObjSrsGridSet = ObjSrsGridSets.Item(1)
...
```

   The method Item returns the individual element at the given index.

5. **Retrieves the group faces of the grid set**

   Now, grid set is available to retrieve GroupFaces.

```
...
   ObjSrsGridSet.GetGroupFaces ObjSrsLastGridFacesAfr,
                               ObjSrsLastGridFacesBfr
...
```

   Here, ObjSrsGridFacesAfterLast is the list of faces which is last face of a group after origin. ObjSrsGridFacesBeforeLast is the list of faces which is last face of a group before origin.

6. **Retrieves a grid face in the grid set**

   In this step UC retrieves a grid face from the list of grid faces

```
...
   Set ObjSrsGridFace = ObjSrsLastGridFacesAfr.Item(1)
...
```

   The method Item returns the individual element at the given index.

7. **Retrieves short name of the grid face**

   In this step UC retrives short name of the obtained grid face.

```
...
    Dim StrShortName As String
    StrShortName = ObjSrsGridFace.ShortName
...
```

   The property ShortName is the short name of the grid face.

8. **Sets short name of the grid face**

```
...
    ObjSrsGridFace.ShortName = StrShortName
...
```

ShortName property is used to set the short name of the grid face.

9. **Retrieve the category on the grid face**

   In this step UC retrieves the category of the grid face from the Category property of the grid face.

```
...
    'Retrieve the category
    Dim strCategory As String
    strCategory = ObjSrsGridFace.Category
...
```

10. **Set a new category on the grid face**

    In this step UC sets a new category on the grid face. Before calling this method ensure that the "Project Dictionary" resource having the category to be assigned is set in the "Common Geometry Resource" table in data setup.

```
...
    'Sets the Category
    strCategory = "WebFrame"
    ObjSrsGridFace.Category = strCategory
...
```

11. **Retrieves the Reference hull surface of the SRS part**

    In this step UC retrieves the hull surface from the part.

```
...
    Dim ObjSrsRefSurface As Reference
    Set ObjSrsRefSurface = ObjSrsPart.ReferenceSurface
...
```

12. **Retrieves the MidShip plane of the SRS part**

    In this step UC retrieves the MidShip plane from the part.

```
...
    Dim ObjSrsMidShip As SrsMidShip
    Set ObjSrsMidShip = ObjSrsPart.MidShip
...
```

13. **Retrieves the CentreLine plane of the SRS part**

    In this step UC retrieves the CentreLine plane from the part.

```
...
    Dim ObjSrsCentreLine As SrsCentreLine
    Set ObjSrsCentreLine = ObjSrsPart.CentreLine
...
```

14. **Updates the Part object**

    Update is called to update the Part object.

```
...
    'Update the Part object
    ObjPart.Update
...
```

# Using Coordinate Converter

This use case primarily focuses on the methodology to use SRS method.

---

Before you begin: Note that:

- You should first launch CATIA and import the `CAASrsCoordinateConverterUc.3dxml` file supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSrsSpaceReferenceSystem\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
- The below need to be set for use Coordinate methods :
  1) Change Unit to M
  2) Attach Resource Table to Collab Space in Data Setup

Related Topics
Space Reference System Object Model Map
Launching an Automation Use Case

Where to find the macro: CAAScdCoordinateConverterSource.htm

This use case can be divided in fourteen steps:

1. Retrieves the active editor of opened product
2. Retrieves Selection object
3. Retrieves Instantiate Service
4. Using Coordinate Converter method(Absolute Coordinate to Srs Coordinate)
5. Using Coordinate Converter method(Srs Coordinate to Absolute Coordinate)

1. **Retrieves the active editor of the Srs product**

   As a first step, the UC retrieves the editor for the Srs product opened in CATIA.

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

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SRSProdSel variable. To retrieve the Selection object `SRSPrdEditor` is used.

```
 ...
  Set SRSProdSel = SRSPrdEditor.Selection
 ...
```

3. **Retrieves Instantiate Service**

   In this step UC retrieves Service to use Coordinate Converter methods.

```
...
  Dim ObjSrsService As SrsInstantiateService
  Set ObjSrsService = CATIA.ActiveEditor.GetService("SrsInstantiateService")

  ObjSrsService.CreateSrsCoordinateConverter ObjSrsCoordinateConverter
...
```

4. **Using Coordinate Converter method(Absolute Coordinate to Srs Coordinate)**

   In this step UC to use methods

```
...
  Dim X, Y, Z As Double         ' Input
  Dim luSRSCoord(2) As Variant  ' Output

  ...

  ObjSrsCoordinateConverter.ConvertAbsoluteCoordToSrsCoord X, Y, Z, luSRSCoord
...
```

   The method Item returns the individual element at the given index.

5. **Using Coordinate Converter method(Srs Coordinate to Absolute Coordinate)**

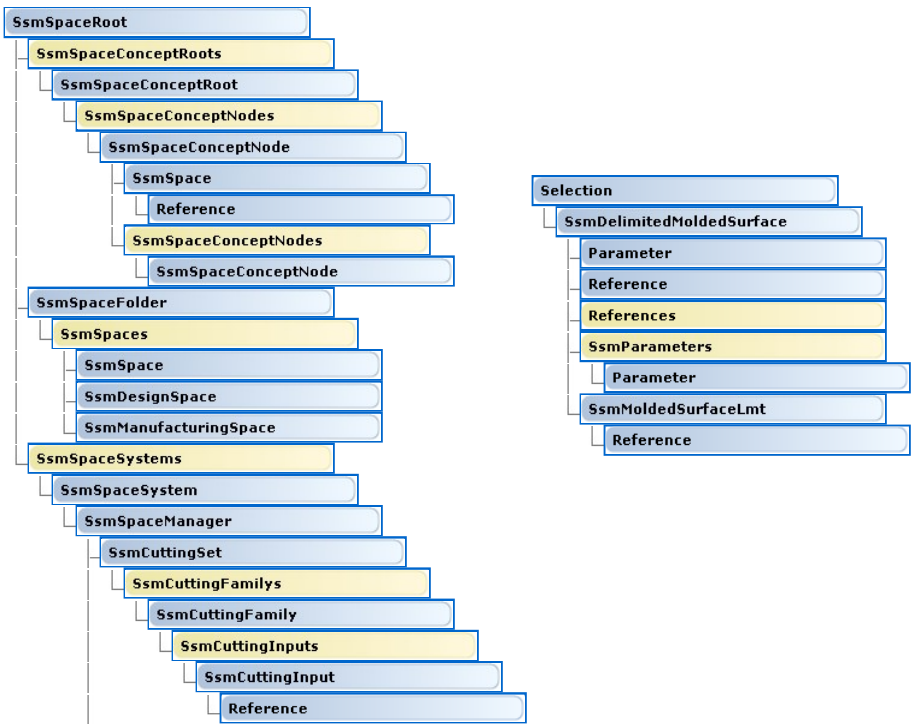
   It's same step.

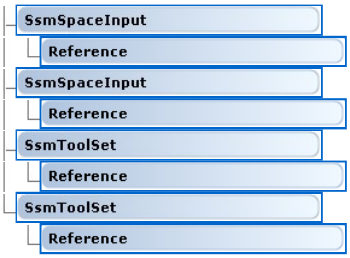
```
...
  Dim luSRSCoord(2) As Variant  ' Input
  Dim X, Y, Z As Double         ' Output

  ...

  ObjSrsCoordinateConverter.ConvertSrsCoordToAbsoluteCoord luSRSCoord, X, Y, Z
...
```

   The method Item returns the individual element at the given index.

# Space Allocation Object Model Map

See Also Legend






# Retrieving Space Allocation Data

This use case primarily focuses on the methodology to retrieve and edit Space Allocation data.

Related Topics
Space Allocation Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- Launch CATIA
- Open a SPL product structure from database or create one before launching the macro. A sample `CAASplUcAssembly.3dxml` file is supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSpaceAllocation\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.
- Keep product editor active before launching the macro

Where to find the macro: CAAScdSpaceAllocationSource.htm

This use case can be divided in 15 steps:

1. Open model in the editor
2. Retrieves active object
3. Retrieves Space Root
4. Retrieves Space Folder and Spaces
5. Retrieves Space System
6. Retrieves Space Manager
7. Retrieves Bounding Box and External Volume
8. Retrieves SpaceCell Set,Cutting Set and family
9. Retrieves Molded Surface
10. Retrieves Molded Surface Internal Attributes
11. Retrieves Molded Surface Limit Internal Attributes
12. Retrieves Space Concept Root
13. Retrieves Space Concept Node
14. Retrieves Space Reference
15. Retrieves Design or Manufacturing Space Reference

1. **User should opens model in the editor**

   As a first step, open a SPL product structure. The supplied sample "CAASplUcAssembly.3dxml" can be imported and open the product named "Root".

2. **Retrieves active Object**

   As a next step, the UC retrieves the active object in oUIActiveObject variable. To retrieve the active object, the active editor **`SSMPrdEditor`** is used.

```
...
 Set SSMPrdEditor = CATIA.ActiveEditor
 Set oUIActiveObject = SSMPrdEditor.ActiveObject
...
```

3. **Retrieves Space Root**

   In this step UC retrieves Space Root Object.

```
...
  Dim objectOccur As VPMOccurrence
  Set objectOccur = oUIActiveObject

  Dim oListChildrenOccurrences As VPMOccurrences
  Set oListChildrenOccurrences = objectOccur.Occurrences

  Dim SsmSpaceRootObj As SsmSpaceRoot

  objectcount = oListChildrenOccurrences.Count
  For j = 1 To objectcount
    Set TempVPMOccur = oListChildrenOccurrences.Item(j)
    Set TempVPMInst = TempVPMOccur.InstanceOccurrenceOf
    Set TempVPMReference = TempVPMInst.ReferenceInstanceOf

    Dim name As String

    name = TempVPMReference.GetCustomType

    If TempVPMReference.GetCustomType = "SPP_SpaceRoot" Then
      Set SsmSpaceRootObj = TempVPMReference
    End If
  Next
...
```

4. **Retrieves Space Folder and Spaces**

   In this step UC retrieves Space Folder.

```
...
 Dim Spacefolder As SsmSpaceFolder
 Set Spacefolder = SsmSpaceRootObj.Spacefolder
```

```
  Dim SpaceList As SsmSpaces
  Set SpaceList = Spacefolder.Spaces
 ...
```

5. **Retrieves Space System**

   In this step UC retrieves Space System.

```
...
  Dim SsmSpaceSys As SsmSpaceSystem
  Dim SpaceSysList As SsmSpaceSystems

  Set SpaceSysList = SsmSpaceRootObj.SpaceSystems

  Set SsmSpaceSys = SpaceSysList.Item(1)
...
```

6. **Retrieves Space Manager**

   Now, retrieve the Space manager from Space System VPM Reference.

```
...

  Dim SsmManager As SsmSpaceManager
  Set SsmManager = SsmSpaceSys.SpaceManager
...
```

7. **Retrieves Bounding Box and External Volume**

   In this step UC retrieves Bounding Box and External Volume from Space Manager.

```
...
  Dim SpaceExtrVol As SsmSpaceInput
  Set SpaceExtrVol = SsmManager.ExternalVolume
  Set extrvolume   = SpaceExtrVol.SpaceElement

  Dim SpaceboundingBox As SsmSpaceInput
  Set SpaceboundingBox = SsmManager.BoundingBox
  Set boundBox         = SpaceboundingBox.SpaceElement
...
```

   These methods will give user the access to the Bounding box and External Volume.

8. **Retrieves SpaceCell Set,Cutting Set and family**

   In this step UC retrieves SpaceCell Set,Cutting Set and family from Space Manager.

```
...
  Dim SpaceCellTool As SsmToolSet
  Dim SpaceCellChild As References
  Set SpaceCellTool = SsmManager.SpaceCellSet
  SpaceCellTool.GetChildren SpaceCellChild

  Dim CuttingCellTool As SsmCuttingSet
  Dim CuttingSetChild As SsmCuttingFamilys
  Dim CuttigFamilychild As SsmCuttingInputs
  Set CuttingCellTool = SsmManager.CuttingSet
  CuttingCellTool.GetChildren CuttingSetChild

  Dim cuttingfamily As SsmCuttingFamily
  Set cuttingfamily = CuttingSetChild.Item(1)

  cuttingfamily.GetChildren CuttigFamilychild

  Dim cuttingelementInput As SsmCuttingInput
  Set cuttingelementInput = CuttigFamilychild.Item(1)

  Set cuttingelement = cuttingelementInput
...
```

   These methods will give user the access to the Space Cells and Cutting elements.

9. **Retrieves Molded Surfaces Molded Surface**

   In this step UC retrives Molded Surface. Same as in Retrieving Space Mananger, User can reach to the part whcih contain Molded Surface

```
...
  Dim oVPMRepInsts As VPMRepInstances
  Set oVPMRepInsts = SpaceSysVPMReference.RepInstances

  Dim oVPMRepInst As VPMRepInstance
  Set oVPMRepInst = oVPMRepInsts.Item(1)

  Dim oVPMRepRef As VPMRepReference
  Set oVPMRepRef = oVPMRepInst.ReferenceInstanceOf

  Dim oPart As Part

  Set oPart = oVPMRepRef.GetItem("Part")
  Dim MoldedSurface As SsmDelimitedMoldedSurface
  Set MoldedSurface = oPart.FindObjectByName("D.2")
...
```

   Using this method, user can reach to the desired molded surface

10. **Retrieves Molded Surface Internal Attributes**

```
...
  Dim Orientation As Long
```

```
  Set MSSupport = MoldedSurface.Support
  Orientation   = MoldedSurface.SupportOrientation
  Set MSOffset  = MoldedSurface.SupportOffset
...
```

11. **Retrieves Molded Surface Limit Internal Attributes**

    Molded Surface Limit Interface can be retrieved from Molded Surface and then it can be used to access internal attributes

```
...
  Dim MoldedSurfaceLmt As SsmMoldedSurfaceLmt
  Set MoldedSurfaceLmt = MoldedSurface

  Orientation = MoldedSurfaceLmt.GetOrientation(index)
  Set limitingObj = MoldedSurfaceLmt.GetLimitingObject(index)
...
```

12. **Retrieves Space Concept Root**

    In this step UC retrieves Space Concept Root.

```
...
  Dim SsmSpaceConRoot As SsmSpaceConceptRoot
  Dim SpaceConRootList As SsmSpaceConceptRoots
  Set SpaceConRootList = SsmSpaceRootObj.SpaceConceptRoots

  Set SsmSpaceConRoot = SpaceConRootList.Item(1)
...
```

    User can get aggregated Space concept nodes from this interface

```
...
  Dim SpaceConNodeList As SsmSpaceConceptNodes
  Set SpaceConNodeList = SsmSpaceConRoot.SpaceConceptNodes
...
```

13. **Retrieves Space Concept Node**

    In this step UC retrieves Space Concept Node

```
...
  Dim Size As Long
  Dim SpaceConNode As SsmSpaceConceptNode

  Size = SpaceConNodeList.Count

  For i = 1 To Size
      Set SpaceConNode = SpaceConNodeList.Item(i)
  Next
...
```

    User can get aggregated Space concept nodes from this interface

```
...
  Dim SpaceConNodeList2 As SsmSpaceConceptNodes
  Set SpaceConNodeList2 = SpaceConNode.SpaceConceptNodes
...
```

14. **Retrieves Space Reference**

    In this step UC retrieves the Space Reference from Space Concept node.

```
...
  Dim SpaceRef As SsmSpace
  Set SpaceRef = SpaceConNode.Space
...
```

    User can then access all its attributes from this Interface

```
...
  Set SpaceRefGeo = SpaceRef.VolumeGeom
  SpaceVolume = SpaceRef.Volume

...
```

15. **Retrieves Design or Manufacturing Space Reference**

    In this step UC retrieves the Design or Manufacturing Space Reference from Space Reference.

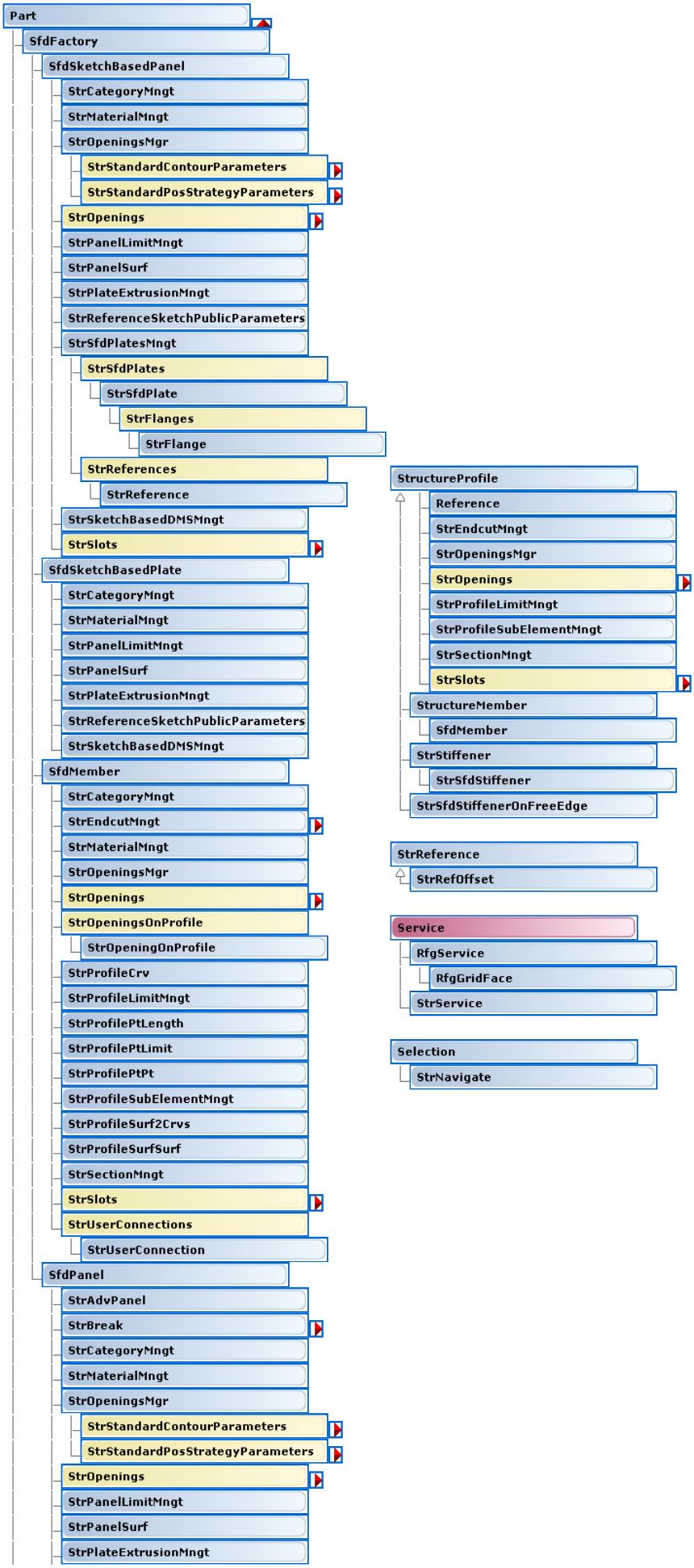
```
...
  Dim SpaceRefDesign As SsmDesignSpace
  Dim SpaceRefManuf  As SsmManufacturingSpace

  Set SpaceRefDesign = SpaceRef

...
```

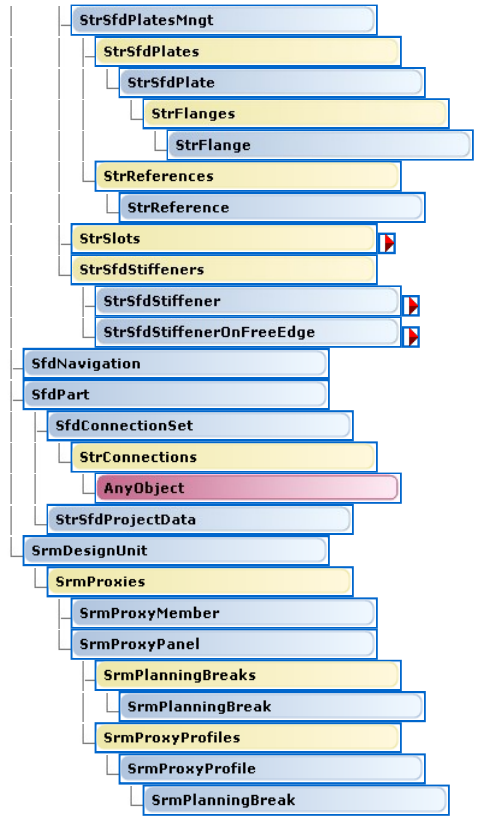
    It can be either Design or Manufacturing Space Reference. User can then access all its attributes from this Interface

```
...

  SpaceFloorArea = SpaceRefDesign.FloorArea

...
```

# Structure Functional Design Object Model Map

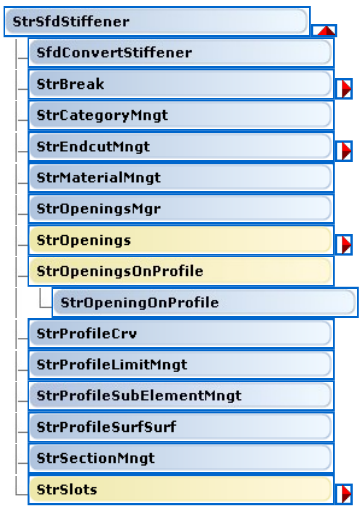
See Also Legend


Part
SfdFactory
SfdSketchBasedPanel
StrCategoryMngt
StrMaterialMngt
StrOpeningsMgr
StrStandardContourParameters
StrStandardPosStrategyParameters
StrOpenings
StrPanelLimitMngt
StrPanelSurf
StrPlateExtrusionMngt
StrReferenceSketchPublicParameters
StrSfdPlatesMngt
StrSfdPlates
StrSfdPlate
StrFlanges
StrFlange
StrReferences
StrReference
StrSketchBasedDMSMngt
StrSlots
SfdSketchBasedPlate
StrCategoryMngt
StrMaterialMngt
StrPanelLimitMngt
StrPanelSurf
StrPlateExtrusionMngt
StrReferenceSketchPublicParameters
StrSketchBasedDMSMngt
SfdMember
StrCategoryMngt
StrEndcutMngt
StrMaterialMngt
StrOpeningsMgr
StrOpenings
StrOpeningsOnProfile
StrOpeningOnProfile
StrProfileCrv
StrProfileLimitMngt
StrProfilePtLength
StrProfilePtLimit
StrProfilePtPt
StrProfileSubElementMngt
StrProfileSurf2Crvs
StrProfileSurfSurf
StrSectionMngt
StrSlots
StrUserConnections
StrUserConnection
SfdPanel
StrAdvPanel
StrBreak
StrCategoryMngt
StrMaterialMngt
StrOpeningsMgr
StrStandardContourParameters
StrStandardPosStrategyParameters
StrOpenings
StrPanelLimitMngt
StrPanelSurf
StrPlateExtrusionMngt
StructureProfile
Reference
StrEndcutMngt
StrOpeningsMgr
StrOpenings
StrProfileLimitMngt
StrProfileSubElementMngt
StrSectionMngt
StrSlots
StructureMember
SfdMember
StrStiffener
StrSfdStiffener
StrSfdStiffenerOnFreeEdge
StrReference
StrRefOffset
Service
RfgService
RfgGridFace
StrService
Selection
StrNavigate



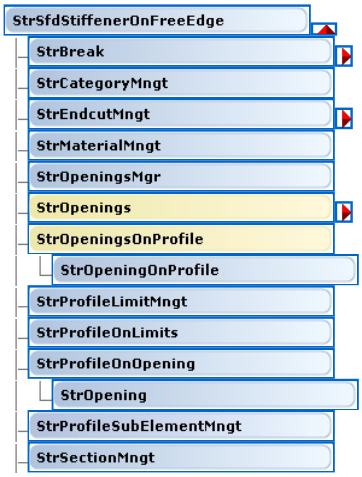

# StrSfdStiffener Object Model Map

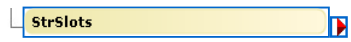
See Also Legend




# StrSfdStiffenerOnFreeEdge Object Model Map

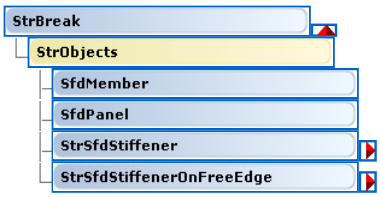
See Also Legend






# StrBreak Object Model Map

See Also Legend

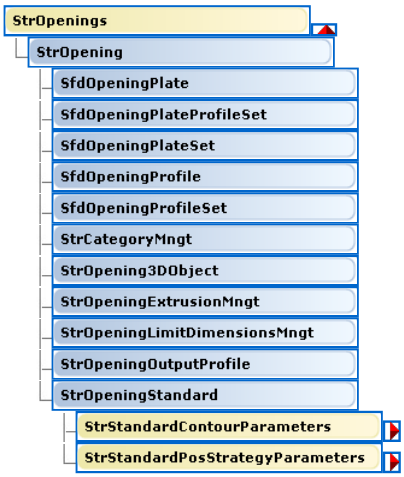



# StrEndcutMngt Object Model Map

See Also Legend

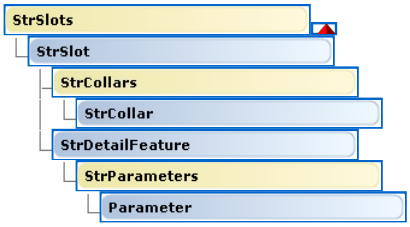

# StrOpenings Object Model Map

See Also Legend



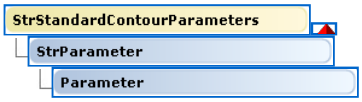

# StrSlots Object Model Map

See Also Legend




# StrStandardContourParameters Object Model Map

See Also Legend




# StrStandardPosStrategyParameters Object Model Map

See Also Legend

# SfdFactory Object

See Also Legend Use Cases Properties Methods


Represents the Structure Functional Design factory object.

This factory object can be retrieved thanks to the **GetCustomerFactory** method of the **Part** object. This example assumes that the **Part** object is the active one.

```
Dim oPart As Part
Set oPart = CATIA.ActiveEditor.ActiveObject
Dim oSfdFact As SfdFactory
Set oSfdFact = oPart.GetCustomerFactory("SfdFactory")
```

## Using the SfdFactory Object

The SfdFactory object enables you to create a member or a panel, and to retrieve a service object, using the following methods:

- **AddMember** method
- **AddPanel** method.

### Creating a Member

A member is created from the **Part** object handled as a **Reference** object.

```
Dim oPartRef As Reference
Set oPartRef = oPart.CreateReferenceFromObject(oPart)
Dim oMember As SfdMember
Set oMember = oSfdFact.AddMember(oPartref)
```

### Creating a Panel

A panel is also is created from the **Part** object handled as a **Reference** object. The factory enables to create basic and advanced plates.

```
Dim oPartRef As Reference
Set oPartRef = oPart.CreateReferenceFromObject(oPart )
Dim oMember As SfdMember
Set oMember = oSfdFact.AddPanel(oPartref, False)
```

Set the second argument of the **AddPanel** method to **False** to create a basic plate, or to **True** to create an advanced plate.

# Creating an SFD Panel

This use case primarily focuses on the methodology to initialize the SFD system and create SFD panel, split panel and break panel.

Related Topics
Structure Functional Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcCGR.3dxml` , `CAAScdSfdUcSR.3dxml` and `CAAScdSfdUcCreatePanel.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.
  To run this use case on server, bind `CAAScdSfdUcCGR.3dxml,CAAScdSfdUcSR.3dxml` PRM settings properly .

Where to find the macro: CAAScdSfdUcCreatePanelSource.htm

This use case can be divided in 12 steps:

1. Searches and opens model which is named as "VbPanel"
2. Retrieves Part object
3. Retrieves Service manager (RfgService)
4. Init resources
5. Initialize SFD system
6. Creates a DeckPanel using DECK.1 as support plane
7. Creates a TransversePanel using CROSS.90 as support plane
8. Splits DeckPanel by using TransversePanel
9. Creates a panel which will be later used for breaking DeckPanel
10. Break the DeckPanel
11. Creates Advanced panel
12. Updates the Part object

1. **Searches and opens model which is named as "VbPanel"**

   As a first step, the UC retrieves a model "VbPanel" from DB and loads it and returns object of the Editor.

   ```
   ...
    Dim SFDPrdEditor  As  Editor
    Dim prdName As String
    prdName = "VbPanel"
    OpenProduct prdName , SFDPrdEditor
   ...
   ```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor is returned in **`SFDPrdEditor`**.

2. **Retrieves Part object**

In this step UC retrieves Part object in `ObjPart` variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

3. **Retrieves Service manager (RfgService)**

   In this step UC retrieves `RfgServices`.

   ```
   ...
     Set Manager = CATIA.ActiveEditor.GetService("RfgServices")
   ...
   ```

   `GetService` method returns `RfgServices`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

4. **Init resources**

   In this step Uc initializes resources for SFD.

   ```
   ...
     Dim ObjSfdFactory As SfdFactory
     Set ObjSfdFactory = ObjPart.GetCustomerFactory("SfdFactory")
     ObjSfdFactory.InitResources
   ...
   ```

   In this step Uc retrieves object `ObjSfdFactory` of type `SfdFactory`. It is retrieved by using `GetCustomerFactory` method. Then object object of type `SfdServices` is retrieved by using method `GetSfdServices`. Then `InitResources` method is called to initialize the SFD resources like molded convention etc.

5. **Initialize SFD system**

   Call `InitializeSFDSystem` method to create reference plane system and hull. `InitializeSFDSystem` method takes a editor as input parameter.

   ```
   ...
     InitializeSFDSystem SFDPrdEditor
   ...
   ```

   The method `InitializeSFDSystem` is detailed as in the below sub steps.

   i. **Creates Reference planes for SFD system**

      ```
      Sub InitializeSFDSystem(iSFDPrdEditor As Editor)
          Manager.CreateProjectData ObjPart
      ...
      ```

      Here `CreateProjectData` method creates reference plane systems which are required for SFD.

   ii. **Retrieves SfdPart and creates Hull**

      ```
      ...
          'Get SfdPart object
          Dim ObjSfdPart As SfdPart
          Set SFDProdSel = iSFDPrdEditor.Selection
          SFDProdSel.Add ObjPart
          Set ObjSfdPart = SFDProdSel.FindObject("CATIASfdPart")
          Set ObjSfdPartRef = ObjPart.CreateReferenceFromObject(ObjSfdPart)
          Set ProjectData = ObjSfdPart.GetProjectData
          Set ProjectDataRef = ObjPart.CreateReferenceFromObject("ProjectData")
          'Create Hull
          Manager.CreateRefSurfaceFeature ObjSfdPartRef, ProjectDataRef
      ...
      ```

      In this step Uc retrieves object of type `SfdPart` in `ObjSfdPart`. To retrieve `ObjSfdPart` add `ObjPart` to selection by using method `Add`. Then call `FindObject` method to get the object `ObjSfdPart` as shown above. reference to the `ObjSfdPart` is created using method `CreateReferenceFromObject` and `GetProjectData` method is called on object `ObjSfdpart` to retrieve the StrSfdProjectData object and reference to the StrSfdProjectData is retrieved in ProjectDataRef using `CreateReferenceFromObject`. After retrieving `ObjSfdPartRef` & `ProjectDataRef` method `CreateRefSurfaceFeature` is called to create the hull.

   iii. **Set this system as SFD system**

      ```
      ...
          Dim oDiagnosis As Long
          Dim oIncorrectDiscName As String
          ObjSfdPart.SetAsSFDSystem oDiagnosis, oIncorrectDiscName
          If oDiagnosis = 2 Or oDiagnosis = 3 Or oDiagnosis = 4 Or oDiagnosis = 5 Then
              Err.Raise 1, Err.Source, "SFD system not set"
              Exit Sub
          End If
      ...
      ```

      This step sets current system as SFD system. Call `SetAsSFDSystem` method to do this. This metohd returns the dignosis first output parameter `oDiagnosis` and system name in `oIncorrectDiscName` if it is already set some other name.

6. **Creates a DeckPanel using DECK.1 as support plane**

   Call `CreatePanel` method to create a panel. `CreatePanel` method returns created panel as output parameter in `ObjSfdPanel`.

   ```
   ...
     Dim ObjSfdPanel As SfdPanel
     CreatePanel ObjSfdPanel
   ...
   ```

   The method `CreatePanel` is detailed as in the below sub steps.

   i. **Creates panel with no properties then set the properties category, support, material and thickness**

```
Sub CreatePanel(oObjSfdPanel As SfdPanel)
    Set ObjPanelSupport = Manager.GetReferencePlane(ObjPart, 1, "DECK.1")
    Set PanelSupport = ObjPart.CreateReferenceFromObject(ObjPanelSupport)
    CreatePanelAndSetData False, "DeckPanel", PanelSupport, "Steel A42", "12mm", 0, oObjSfdPanel
...
```

In this step Uc first retrieves a reference to the plane DECK.1 in `PanelSupport`. For this GetReferencePlane method is used, it takes 3 input parameter first is part object, second defines which plane system the plane belongs to(1=DECK, 2=CROSS, 3=LONG) and third is plane name. Then it calls a method `CreatePanelAndSetData` which creates a empty panel and sets some of the properties of the panel and return the created panel in oObjSfdPanel.

- First parameter defines whether it is a advanced plate or not.
- Second parameter is category of the panel.
- Third parameter is support of the panel.
- Fourth parameter is material of the panel.
- Fifth parameter is thickness of the panel.
- Sixth parameter is throw orientation of the panel.

ii. **Retrieves StrPanelLimitMngt and sets panel limits**

```
...
    'Get StrPanelLimitMngt object
    Dim ObjStrPanelLimitMngt As StrPanelLimitMngt
    Set ObjStrPanelLimitMngt = oObjSfdPanel.StrPanelLimitMngt
    'set limits
    Set ObjLimit1 = Manager.GetReferencePlane(ObjPart, 2, "CROSS.60")
    Set Limit1 = ObjPart.CreateReferenceFromObject(ObjLimit1)
    ObjStrPanelLimitMngt.SetLimitingObject Limit1, -1, 0, 2
    Set ObjLimit2 = Manager.GetReferencePlane(ObjPart, 2, "CROSS.120")
    Set Limit2 = ObjPart.CreateReferenceFromObject(ObjLimit2)
    ObjStrPanelLimitMngt.SetLimitingObject Limit2, -1, 0, 2
    Set ObjLimit3 = Manager.GetReferencePlane(ObjPart, 3, "LONG.-18")
    Set Limit3 = ObjPart.CreateReferenceFromObject(ObjLimit3)
    ObjStrPanelLimitMngt.SetLimitingObject Limit3, -1, 0, 2
    Set ObjLimit4 = Manager.GetReferencePlane(ObjPart, 3, "LONG.18")
    Set Limit4 = ObjPart.CreateReferenceFromObject(ObjLimit4)
    ObjStrPanelLimitMngt.SetLimitingObject Limit4, -1, 0, 2
...
```

In this step Uc retrieves the object `ObjStrPanelLimitMngt` of type `StrPanelLimitMngt`. To set the limits of the panel, Uc first retrieves the reference of the plane and set it as limiting object of the panel by using method `SetLimitingObject`. Method `SetLimitingObject` takes 3 input parameters. First parameter is reference of the limiting plane, second parameter defines the index of the limit(-1 means new limit is added, if LimitIndex>0 means limit is set at this position). And third parameter is orientation of limit, here 0 is used means orientation is automatically computed. We advice you to use 0 always.

iii. **Calls method Run to complete the creation panel**

```
...
    Run oObjSfdPanel
...
```

Method `Run` generates plates under panel. This method takes panel object as input parameter.

iv. **Updates created panel object**

```
...
    ObjPart.UpdateObject oObjSfdPanel
End Sub
```

Method `UpdateObject` updates the created panel.

7. **Creates a TransversePanel using CROSS.90 as support plane**

Call `CreateTransversalPanel` method to create a transversal panel. `CreateTransversalPanel` method returns created panel as output parameter in `ObjTransvSfdPanel`.

```
...
  Dim ObjTransvSfdPanel As SfdPanel
  CreateTransversalPanel ObjTransvSfdPanel
...
```

The method `CreateTransversalPanel` is detailed as in the below sub steps.

i. **Creates panel with no properties then set the properties category, support, material and thickness**

```
Sub CreateTransversalPanel(oObjSfdPanel As SfdPanel)
    Set ObjPanelSupport = Manager.GetReferencePlane(ObjPart, 2, "CROSS.90")
    Set PanelSupport = ObjPart.CreateReferenceFromObject(ObjPanelSupport)
    CreatePanelAndSetData False, "TransversePanel", PanelSupport, "Steel A42", "12mm", 0, oObjSfdPanel
...
```

In this step Uc first retrieves a reference to the plane CROSS.90 in `PanelSupport`. Then it calls a method `CreatePanelAndSetData` which creates a empty panel and sets some of the properties of the panel and return the created panel in oObjSfdPanel. Here second parameter is "TransversePanel", means the panel created is transversal panel.

ii. **Retrieves StrPanelLimitMngt and sets panel limits**

```
...
    'Get StrPanelLimitMngt object
    Dim ObjStrPanelLimitMngt As StrPanelLimitMngt
    Set ObjStrPanelLimitMngt = oObjSfdPanel.StrPanelLimitMngt
    'set limits
    Set ObjLimit1 = Manager.GetReferencePlane(ObjPart, 1, "DECK.6")
    Set Limit1 = ObjPart.CreateReferenceFromObject(ObjLimit1)
    ObjStrPanelLimitMngt.SetLimitingObject Limit1, -1, 0, 2
    ObjStrPanelLimitMngt.InvertLimit -1
    Set ObjLimit2 = Manager.GetReferencePlane(ObjPart, 3, "LONG.-18")
    Set Limit2 = ObjPart.CreateReferenceFromObject(ObjLimit2)
    ObjStrPanelLimitMngt.SetLimitingObject Limit2, -1, 0, 2
    Set ObjLimit3 = Manager.GetReferencePlane(ObjPart, 3, "LONG.18")
    Set Limit3 = ObjPart.CreateReferenceFromObject(ObjLimit3)
```

```
    ObjStrPanelLimitMngt.SetLimitingObject Limit3, -1, 0, 2
    Set ObjLimit4 = Manager.GetReferencePlane(ObjPart, 1, "DECK.0")
    Set Limit4 = ObjPart.CreateReferenceFromObject(ObjLimit4)
    ObjStrPanelLimitMngt.SetLimitingObject Limit4, -1, 0, 2
...
```

In this step Uc retrieves the object `ObjStrPanelLimitMngt` of type `StrPanelLimitMngt`. To set the limits of the panel, Uc first retrieves the reference of the plane and set it as limiting object of the panel by using method `SetLimitingObject`. Method `SetLimitingObject` takes 3 input parameters. First parameter is reference of the limiting plane, second parameter defines the index of the limit(-1 means new limit is added, if LimitIndex>0 means limit is set at this position). And third parameter is orientation of limit, here 0 is used means orientation is automatically computed. We advice you to use 0 always.

iii. **Retrieves StrPlateExtrusionMngt and reverse ThrowOrientation**

```
...
    Dim ObjStrPlateExtrusionMngt As StrPlateExtrusionMngt
    Set ObjStrPlateExtrusionMngt = oObjSfdPanel.StrPlateExtrusionMngt
    ObjStrPlateExtrusionMngt.ReverseThrowOrientation
...
```

This step is added to introduce you to the method `ReverseThrowOrientation`, it is used to invert the ThrowOrienation of the panel.

iv. **Calls method Run to complete the creation panel**

```
...
    Run oObjSfdPanel
...
```

Method `Run` generates plates under panel. This method takes panel object as input parameter.

v. **Updates created panel object**

```
...
    ObjPart.UpdateObject oObjSfdPanel
End Sub
```

Method `UpdateObject` updates the created panel.

8. **Splits DeckPanel by using TransversePanel**

Call `CreateSplitPlates` method to split a panel by using another panel. `CreateSplitPlates` method takes two panel object as input parameter, first input parameter is the panel which will be split and second input parameter is the panel using which panel will be split.

```
...
  CreateSplitPlates ObjSfdPanel, ObjTransvSfdPanel
...
```

The method `CreateSplitPlates` is detailed as in the below sub steps.

i. **Retrieves StrSfdPlatesMngt object**

```
Sub CreateSplitPlates(iObjSfdPanel As SfdPanel, iCuttingPanel As SfdPanel)
    Dim ObjSfdPlatesMngt As StrSfdPlatesMngt
    Set ObjSfdPlatesMngt = iObjSfdPanel.StrSfdPlatesMngt
...
```

In this step Uc retrieves object `ObjSfdPlatesMngt` of type `StrSfdPlatesMngt`.

ii. **Retrieves CuttingElements**

```
...
    Dim ListOfCuttingRefs As StrReferences
    Set ListOfCuttingRefs = ObjSfdPlatesMngt.CuttingElements
...
```

Here Uc retrieves object `ListOfCuttingRefs`, which is of type `StrReferences`. `StrReferences` is a collection object which holds Reference type objects in it. It provides methods such as Add(to add an reference in the collection), Item(to retrieve a particular reference from the collection). These `CuttingElements` will be used to split the panel.

iii. **Adds the references of cutting elements in the list**

```
...
    Set CuttingElemRef = ObjPart.CreateReferenceFromObject(iCuttingPanel)
    ListOfCuttingRefs.Add CuttingElemRef
...
```

Here Uc creates reference to the `iCuttingPanel` and adds it to the `ListOfCuttingRefs` by using method `Add`. Multiple panels can be added to this collection using `Add` method.

iv. **Sets the list of cutting elements**

```
...
    ObjSfdPlatesMngt.CuttingElements = ListOfCuttingRefs
...
```

Set the list of cutting elements to split the panel to the property `CuttingElements`.

v. **Calls method Run to create split plates**

```
...
    ObjSfdPlatesMngt.Run
...
```

Method `Run` generates plates under panel i.e. result after splitting panel.

vi. **Updates created panel object**

```
...
    ObjPart.UpdateObject iObjSfdPanel
End Sub
```

Method `UpdateObject` updates the created panel.

9. **Creates a panel which will be later used for breaking DeckPanel**

Call `CreateSplitPlates` method to split a panel by using another panel. `CreateSplitPlates` method takes two panel object as input parameter, first input parameter is the panel which will be split and second input parameter is the panel using which panel will be split.

```
...
  Dim SplittingPanel As SfdPanel
  CreatePanelForBreak SplittingPanel
...
```

This method creates a panel with support CROSS.115 and limits are DECK.2, LONG.-18, LONG.18, DECK.0. Created panel will returned as output parameter in `SplittingPanel`. Later this panel is used for breaking DeckPanel created above. To see the detailed steps of this method go to CAAScdSfdUcCreatePanelSource.htm.

10. **Break the DeckPanel**

Call `BreakPlate` method to break a panel by using another panel. `BreakPlate` method takes two panel object as input parameters, first input parameter is the panel which will be broken and second input parameter is the panel using which panel will be broken.

```
...
  BreakPlate ObjSfdPanel, SplittingPanel
...
```

The method `CreateSplitPlates` is detailed as in the below sub steps.

i. **Retrieves StrBreak object**

```
Sub BreakPlate(iPanelToBreak As SfdPanel, iSplittingPanel As SfdPanel)
    Dim ObjStrBreak As StrBreak
    Set ObjStrBreak = iPanelToBreak.StrBreak
...
```

In this step Uc retrieves object `ObjStrBreak` of type `StrBreak`.

ii. **Retrieves SplittingElements**

```
...
    Dim CollOfSplitRef As StrReferences
    Set CollOfSplitRef = ObjStrBreak.SplittingElements
...
```

Here Uc retrieves object `CollOfSplitRef`, which is of type `StrReferences`. `StrReferences` is a collection object which holds Reference type objects in it. It provides methods such as Add(to add an reference in the collection), Item(to retrieve a particular reference from the collection). These `SplittingElements` will be used to break the panel.

iii. **Creates reference to the splitting panel and adds it in the SplittingElements list**

```
...
    Set SplittingElemRef = ObjPart.CreateReferenceFromObject(iSplittingPanel)
    CollOfSplitRef.Add SplittingElemRef
...
```

Here Uc creates reference to the `iSplittingPanel` and adds it to the `CollOfSplitRef` by using method `Add`. Multiple panels can be added to this collection using `Add` method.

iv. **Break the panel**

```
...
    ListOfResults = ObjStrBreak.Break(CollOfSplitRef)
...
```

Call method `Break` to break the panel. Method `Break` takes list of objects using which panel will be broken as input parameter and it returns the resultant broken panel list.

v. **Calls method Run to complete the creation panel**

```
...
    Run iPanelToBreak
...
```

Method `Run` generates plates under panel i.e. result after breaking panel.

vi. **Updates created panel object**

```
...
    ObjPart.UpdateObject iPanelToBreak
End Sub
```

Method `UpdateObject` updates the created panel.

11. **Creates Advanced panel**

Call `CreateAdvancedPanel` method to create a advanced panel. `CreateAdvancedPanel` method returns created panel as output parameter in `ObjAdvancedPanel`.

```
...
  Dim ObjAdvancedPanel As SfdPanel
  CreateAdvancedPanel ObjAdvancedPanel
```

...

The method `CreateAdvancedPanel` is detailed as in the below sub steps.

i. **Creates panel with no properties then set the properties category, support, material and thickness**

```
Sub CreateAdvancedPanel(oObjAdvancedPanel As SfdPanel)
    Set ObjPanelSupport = Manager.GetReferencePlane(ObjPart, 2, "CROSS.0")
    Set PanelSupport = ObjPart.CreateReferenceFromObject(ObjPanelSupport)
    CreatePanelAndSetData True, "SldPanel", PanelSupport, "Steel A42", "12mm", 0, oObjAdvancedPanel
...
```

In this step Uc first retrieves a reference to the plane CROSS.0 in `PanelSupport`. Then it calls a method `CreatePanelAndSetData` which creates a empty panel and sets some of the properties of the panel and return the created panel in oObjAdvancedPanel. Here first parameter is True, means the panel created is advanced panel.

ii. **Retrieves StrPanelLimitMngt and sets panel limits**

```
...
    'Get StrPanelLimitMngt object
    Dim ObjStrPanelLimitMngt As StrPanelLimitMngt
    Set ObjStrPanelLimitMngt = oObjAdvancedPanel.StrPanelLimitMngt
    'set limits
    Set ObjLimit1 = Manager.GetReferencePlane(ObjPart, 1, "DECK.8")
    Set Limit1 = ObjPart.CreateReferenceFromObject(ObjLimit1)
    ObjStrPanelLimitMngt.SetLimitingObject Limit1, -1, 0, 2
    ObjStrPanelLimitMngt.InvertLimit -1
    Set ObjLimit2 = Manager.GetReferencePlane(ObjPart, 3, "LONG.10")
    Set Limit2 = ObjPart.CreateReferenceFromObject(ObjLimit2)
    ObjStrPanelLimitMngt.SetLimitingObject Limit2, -1, 0, 2
    ObjStrPanelLimitMngt.InvertLimit -1
    Set ObjLimit3 = Manager.GetReferencePlane(ObjPart, 1, "DECK.3")
    Set Limit3 = ObjPart.CreateReferenceFromObject(ObjLimit3)
    ObjStrPanelLimitMngt.SetLimitingObject Limit3, -1, 0, 2
    ObjStrPanelLimitMngt.InvertLimit -1
    Set ObjLimit4 = Manager.GetReferencePlane(ObjPart, 3, "LONG.16")
    Set Limit4 = ObjPart.CreateReferenceFromObject(ObjLimit4)
    ObjStrPanelLimitMngt.SetLimitingObject Limit4, -1, 0, 2
    ObjStrPanelLimitMngt.InvertLimit -1
    Set ObjLimit5 = Manager.GetReferencePlane(ObjPart, 1, "DECK.1")
    Set Limit5 = ObjPart.CreateReferenceFromObject(ObjLimit5)
    ObjStrPanelLimitMngt.SetLimitingObject Limit5, -1, 0, 2
    Set ObjLimit6 = Manager.GetReferencePlane(ObjPart, 3, "LONG.-17")
    Set Limit6 = ObjPart.CreateReferenceFromObject(ObjLimit6)
    ObjStrPanelLimitMngt.SetLimitingObject Limit6, -1, 0, 2
...
```

In this step Uc retrieves the object `ObjStrPanelLimitMngt` of type `StrPanelLimitMngt`. To set the limits of the panel, Uc first retrieves the reference of the plane and set it as limiting object of the panel by using method `SetLimitingObject`. Method `SetLimitingObject` takes 3 input parameters. First parameter is reference of the limiting plane, second parameter defines the index of the limit(-1 means new limit is added, if LimitIndex>0 means limit is set at this position). And third parameter is orientation of limit, here 0 is used means orientation is automatically computed. We advice you to use 0 always. At some places limit orientation is inverted by using method `Invert` as per requirement.

iii. **Sets the 3rd limit as the last limit of the panel**

User can also change limit oredering which shown as follows:

```
...
    Dim ObjStrAdvPanel As StrAdvPanel
    Set ObjStrAdvPanel = oObjAdvancedPanel.StrAdvPanel
    ObjStrAdvPanel.SetAsLastLimit 3
...
```

Here Uc retrieves the object `StrAdvPanel` which is of type `StrAdvPanel`. `SetAsLastLimit` method is called to set a particular limit as last limit in the limits list. This method takes index of the limit as input parameter which will be moved to the last.

iv. **Calls method Run to complete the creation panel**

```
...
    Run oObjAdvancedPanel
...
```

Method `Run` generates plates under panel. This method takes panel object as input parameter.

v. **Updates created panel object**

```
...
    ObjPart.UpdateObject oObjAdvancedPanel
End Sub
```

Method `UpdateObject` updates the created advanced panel.

12. **Updates the Part object**

```
...
    ObjPart.Update
End Sub
```

`Update` method updates the `ObjPart`

**Detailed steps of methods called in Uc**

- **CreatePanelAndSetData method**

This method creates a new empty panel, sets some properties of the panel and returns the created panel in `oObjSfdPanel` object. Properties which will be set by this method are category, support, material, plate geometry parameters of the panel.

```
Sub CreatePanelAndSetData(iIsAdvPanel As Boolean, iCategory As String, iSupport As Reference, iMaterial As String,
                          iThickness As String, iThrowOrientation As Long, oObjSfdPanel As SfdPanel)

'   1- Retrieves SfdFactory and creates a new empty panel
    'Get SfdFactory object
    Dim ObjSfdFactory As SfdFactory
    Set ObjSfdFactory = ObjPart.GetCustomerFactory("SfdFactory")
    'Create a new empty panel(with no data set)
    Set DestPart = ObjPart.CreateReferenceFromObject(ObjPart)
    Set oObjSfdPanel = ObjSfdFactory.AddPanel(DestPart, iIsAdvPanel)

'   2- Retrieves StrCategoryMngt and sets category
    'Get StrCategoryMngt object
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = oObjSfdPanel.StrCategoryMngt
    'Set Category and automatic naming true
    ObjStrCategoryMngt.SetCategory iCategory
    ObjStrCategoryMngt.AutomaticName = True

'   3- Retrieves StrPanelSurf and sets support of the panel
    'Get StrPanelSurf object
    Dim ObjSfdPanelSupport As StrPanelSurf
    Set ObjSfdPanelSupport = oObjSfdPanel.StrPanelSurf
    'Set support
    ObjSfdPanelSupport.Support = iSupport

'   4- Retrieves StrMaterialMngt and sets material
    'Get StrMaterialMngt object
    Dim ObjStrMaterialMngt As StrMaterialMngt
    Set ObjStrMaterialMngt = oObjSfdPanel.StrMaterialMngt
    'Set material
    ObjStrMaterialMngt.SetMaterial iMaterial

'   5- Retrieves StrPlateExtrusionMngt and sets plate geometry parameters such as thickness, throw orientation
    'Get StrPlateExtrusionMngt object
    Dim ObjStrPlateExtrusionMngt As StrPlateExtrusionMngt
    Set ObjStrPlateExtrusionMngt = oObjSfdPanel.StrPlateExtrusionMngt
    'Set thickness
    Set ThicknessParm = ObjStrPlateExtrusionMngt.GetThickness
    ThicknessParm.ValuateFromString iThickness
    'Set ThrowOrientation
    ObjStrPlateExtrusionMngt.ThrowOrientation = iThrowOrientation
    'Set orientation mode
    ObjStrPlateExtrusionMngt.OffsetMode = 0
End Sub
```

In the first step of this method `SfdFactory` is retrieved in `ObjSfdFactory` by using method `GetCustomerFactory` from `ObjPart`. Then it calls method `AddPanel` to create a new empty panel and stores it in `oObjSfdPanel` which is a output parameter of this method. Method `AddPanel`takes refererence to the destination part and a flag which defines whether it is a advanced plate or not as input parameters.

In the second step, it retrieves the object of type `StrCategoryMngt` in `ObjStrCategoryMngt` and calls `SetCategory` method to set the category.

In the third step, it retrieves the object of type `StrPanelSurf` in `ObjStrPanelSurf` and sets `Support` property with `iSupport`. This is the support of the panel.

In the fourth step, it retrieves the object of type `StrMaterialMngt` in `ObjStrMaterialMngt` and calls `SetMaterial` method to set the material.

In the fifth step, it retrieves the object of type `StrPlateExtrusionMngt` in `ObjStrPlateExtrusionMngt` and retrieves `ThicknessParm` by using method `GetThickness` method and set the thickness value by using method `ValuateFromString`. It also sets its `ThrowOrientation` and `ThrowOrientation` properties.

- **Run method**

This method generates plates under panel. It takes panel object as input parameter.

```
Sub Run(iObjSfdPanel As SfdPanel)
    'Get StrSfdPlatesMngt object
    Dim ObjSfdPlatesMngt As StrSfdPlatesMngt
    Set ObjSfdPlatesMngt = iObjSfdPanel.StrSfdPlatesMngt
    'Call Run method to complete the creation of plate
    ObjSfdPlatesMngt.Run
End Sub
```

IIn the this method `StrSfdPlatesMngt` is retrieved in `ObjSfdPlatesMngt`. Then it calls method `Run` to generate the plates under panel.

Fig.1: Panel and Advanced Panel


# Creating a SFD Member

This use case primarily focuses on the methodology to create a SFD member.

Related Topics
Structure Functional Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcPart.3dxml`, `CAAScdSfdUcCGR.3dxml`, and `CAAScdSfdUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: CAAScdSfdUcCreateMemberSource.htm

This use case can be divided in 11 steps:

1. Searches and opens model which is named as "SFD_VB_TEST"
2. Retrieves Part object
3. Retrieves Service Manger (RfgService)
4. Creates a Geometrical Set in Part for creating Members in it
5. Creates a member using two surfaces
6. Creates a member using a point and a limit
7. Creates a member using a point and length
8. Creates a member using a curve
9. Creates a member using a surface and two members(crvs)
10. Creates a member using two points
11. Updates the Part object

1. **Searches and opens model which is named as "SFD_VB_TEST"**

   As a first step, the UC retrieves a model "SFD_VB_TEST" from DB and loads it and returns object of the Editor.

   ```
   ...
    Dim SFDPrdEditor  As  Editor
    Dim prdName As String
    prdName = "SFD_VB_TEST"
    OpenProduct prdName , SFDPrdEditor
   ...
   ```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor i
   **`SFDPrdEditor`**.

2. **Retrieves Part object**

   In this step UC retrieves Part object in `ObjPart` variable.

   ```
   ...
     Set ObjPart = SFDPrdEditor.ActiveObject
   ...
   ```

3. **Retrieves Service manager (RfgService)**

   In this step UC retrieves `RfgService`.

   ```
   ...
     Set Manager = CATIA.ActiveEditor.GetService("RfgService")
   ...
   ```

   `GetService` method returns `RfgService`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

4. **Creates a Geometrical Set in Part for creating Members in it**

   In this step Uc creates a geometrical set. The created members (beams) will be aggregated under this.

```
...
  Set oHybridBodies = ObjPart.HybridBodies
  Set oGeometricalSet = oHybridBodies.Add()
  Set RefGeometricalSet = ObjPart.CreateReferenceFromObject(oGeometricalSet)
...
```

In this step Uc creates a geometrical set by retrieving `HybridBodies` object and calling `Add` method on it. Then it creates reference to the created geometrical set a `RefGeometricalSet`.

5. **Creates a member using two surfaces**

Call `CreateMemberSurfSurf` method to create member using two surfaces. `CreateMemberSurfSurf` method takes a reference to the geometrical set as input para created member as output parameter in `ObjMemberSurfSurf`.

```
...
  Dim ObjMemberSurfSurf As SfdMember
  CreateMemberSurfSurf RefGeometricalSet, ObjMemberSurfSurf
...
```

The method `CreateMemberSurfSurf` is detailed as in the below sub steps.

i. **Creates a empty member and sets category, material, profile type and section parameters**

```
Sub CreateMemberSurfSurf(iRefGeometricalSet As Reference, oObjMember As SfdMember)
  CreateMemberAndSetData iRefGeometricalSet, "SldMember", catStrProfileModeSurfSurf, "Steel A42", "WT18x179.5", "catStrTopCen
...
```

In this step Uc calls a method `CreateMemberAndSetData` which creates a empty member under given GeometricalSet sets some of the properties of the mer created member in oObjMember. This method sets category, profile type, material, section name and anchor point of the member. Here iRefGeometricalSet catStrProfileModeSurfSurf is the profile type(it means profile is created on intersection of two surfaces), "Steel A42" is material name, "WT18x179.5" is se "catStrTopCenter" is anchor point.

ii. **Retrieves StrProfileLimitMngt and sets profile limits**

```
...
    Dim ObjStrProfileLimitMngt As StrProfileLimitMngt
    Set ObjStrProfileLimitMngt = oObjMember.StrProfileLimitMngt
    'set Start limit
    Set ObjStartLimit = Manager.GetReferencePlane(ObjPart, 1, "DECK.8")
    Set StartLimit = ObjPart.CreateReferenceFromObject(ObjStartLimit)
    ObjStrProfileLimitMngt.SetLimitingObject 1, StartLimit
    'set End limit
    Set ObjEndLimit = Manager.GetReferencePlane(ObjPart, 1, "DECK.3")
    Set EndLimit = ObjPart.CreateReferenceFromObject(ObjEndLimit)
    ObjStrProfileLimitMngt.SetLimitingObject 2, EndLimit
...
```

In this step Uc retrieves object of type `StrProfileLimitMngt` in `ObjStrProfileLimitMngt`. Reference to the plane DECK.8 is retrieved in `StartLimit` by `GetReferencePlane`. Method `GetReferencePlane` takes 3 input parameters. First parameter is the part from which plane is, second is the index of the plan 1, CROSS = 2, LONG = 3) and third is the name of the plane.

To set start limit, `SetLimitingObject` method is called. It takes two inputs, **1** and **StartLimit**. here 1 means the limit is for Start of profile. In the same mar DECK.3 is set as the end limit for the profile. To set the End limit `SetLimitingObject` method takes first parameter as **2**.

iii. **Retrieves StrProfileSurfSurf and sets surfaces for member creation**

```
...
    'Get StrProfileSurfSurf object
    Dim ObjStrProfileSurfSurf As StrProfileSurfSurf
    Set ObjStrProfileSurfSurf = oObjMember.StrProfileSurfSurf
    'Set FirstSurface
    Set ObjRefFirstSurf = Manager.GetReferencePlane(ObjPart, 2, "CROSS.8")
    Set RefFirstSurf = ObjPart.CreateReferenceFromObject(ObjRefFirstSurf)
    ObjStrProfileSurfSurf.FirstSurface = RefFirstSurf
    'Set SecondSurface
    Set ObjRefSecondSurf = Manager.GetReferencePlane(ObjPart, 3, "LONG.-4")
    Set RefSecondSurf = ObjPart.CreateReferenceFromObject(ObjRefSecondSurf)
    ObjStrProfileSurfSurf.SecondSurface = RefSecondSurf
...
```

This step sets two surfaces FirstSurface and SecondSurface. Member will be created at the intersection of these two surfaces.

Uc first retrieves the object of type `StrProfileSurfSurf`.To set the FirstSurface, it retrieves the reference to the CROSS.8 plane in `RefFirstSurf` and assi `FirstSurface` property of the `ObjStrProfileSurfSurf` object. To set the SecondSurface, it retrieves the reference to the LONG.-4 plane in `RefSecondSu` the `SecondSurface` property of the `ObjStrProfileSurfSurf` object.

iv. **Updates created member object**

```
...
    ObjPart.UpdateObject oObjMember
End Sub
```

Method `UpdateObject` updates the created member.

6. **Creates a member using a point and a limit**

Call `CreateMemberPtLimit` method to create member using a point and a limit.

`CreateMemberPtLimit` method takes a reference to the geometrical set as input parameter and it returns created member as output parameter in `ObjMemberPtLim`

```
...
  Dim ObjMemberPtLimit As SfdMember
  CreateMemberPtLimit RefGeometricalSet, ObjMemberPtLimit
...
```

The method `CreateMemberPtLimit` is detailed as in the below sub steps.

i. **Creates a empty member and sets category, material, profile type and section parameters**

```
Sub CreateMemberPtLimit(iRefGeometricalSet As Reference, oObjMember As SfdMember)
  CreateMemberAndSetData iRefGeometricalSet, "SldMember", catStrProfileModePtLimit, "Steel A42", "WT18x179.5", "catStrTopCent
...
```

In this step Uc calls a method `CreateMemberAndSetData` which creates a empty member under given GeometricalSet sets some of the properties of the mer created member in oObjMember. Third parameter of this method is catStrProfileModePtLimit. Here catStrProfileModePtLimit means profile is created usin

ii. **Creates a point. Later this point will be used for member creation**

```
...
    Set ObjHybridShapeFactory = ObjPart.HybridShapeFactory
    Set ObjHybridShapePointCoord = ObjHybridShapeFactory.AddNewPointCoord(5000, 5000, 0)
    Set RefPoint = ObjPart.CreateReferenceFromObject(ObjHybridShapePointCoord)
...
```

In this step Uc creates a point using method `AddNewPointCoord` and reference to this point is stored in `RefPoint`.

iii. **Retrieves StrProfilePtLimit and sets point and limit and direction**

```
...
    'Get StrProfilePtLimit object
    Dim ObjStrProfilePtLimit As StrProfilePtLimit
    Set ObjStrProfilePtLimit = oObjMember.StrProfilePtLimit
    'set point
    ObjStrProfilePtLimit.StartPoint = RefPoint
    'set limit
    Set ObjMemberLimitRef = Manager.GetReferencePlane(ObjPart, 1, "DECK.4")
    Set MemberLimitRef = ObjPart.CreateReferenceFromObject(ObjMemberLimitRef)
    ObjStrProfilePtLimit.UpToLimit = MemberLimitRef
    'set direction
    ObjStrProfilePtLimit.Direction = MemberLimitRef
...
```

In this step object of type `StrProfilePtLimit` is retrieved in `ObjStrProfilePtLimit`. Then `RefPoint`, which is created in the step above is assigned to th property. Point is set, now Uc needs to set the limit. To set the limit it retrieves the reference to plane DECK.4 in `MemberLimitRef` and assigns it to the `UpT` Same plane is assigned to the `Direction` property also.

iv. **Updates created member object**

```
...
    ObjPart.UpdateObject oObjMember
End Sub
```

Method `UpdateObject` updates the created member.

7. **Creates a member using a point and length**

Call `CreateMemberPtLength` method to create member using a point and length. `CreateMemberPtLength` method takes a reference to the geometrical set as inpu returns created member as output parameter in `ObjMemberPtLength`.

```
...
  Dim ObjMemberPtLength As SfdMember
  CreateMemberPtLength RefGeometricalSet, ObjMemberPtLength
...
```

The method `CreateMemberPtLength` is detailed as in the below sub steps.

i. **Creates a empty member and sets category, material, profile type and section parameters**

```
Sub CreateMemberPtLength(iRefGeometricalSet As Reference, oObjMember As SfdMember)
  CreateMemberAndSetData iRefGeometricalSet, "SldMember", catStrProfileModePtLength, "Steel A42", "WT18x179.5", "catStrTopCen
...
```

In this step Uc calls a method `CreateMemberAndSetData` which creates a empty member under given GeometricalSet and sets some of the properties of the the created member in oObjMember. Third parameter of this method is catStrProfileModePtLength. Here catStrProfileModePtLength means profile is creat length.

ii. **Creates a point. Later this point will be used for member creation**

```
...
    Set ObjHybridShapeFactory = ObjPart.HybridShapeFactory
    Set ObjHybridShapePointCoord = ObjHybridShapeFactory.AddNewPointCoord(20000, 20000, 0)
    Set RefPoint = ObjPart.CreateReferenceFromObject(ObjHybridShapePointCoord)
...
```

In this step Uc creates a point using method `AddNewPointCoord` and reference to this point is stored in `RefPoint`.

iii. **Retrieves StrProfilePtLength and sets point and direction and length**

```
...
    'Get StrProfilePtLength object
    Dim ObjStrProfilePtLength As StrProfilePtLength
    Set ObjStrProfilePtLength = oObjMember.StrProfilePtLength
    'set point
    ObjStrProfilePtLength.StartPoint = RefPoint
    'set direction
    Set ObjRefMemberDirection = Manager.GetReferencePlane(ObjPart, 1, "DECK.7")
    Set RefMemberDirection = ObjPart.CreateReferenceFromObject(ObjRefMemberDirection)
    ObjStrProfilePtLength.Direction = RefMemberDirection
    'set length
    Set LengthParm = ObjStrProfilePtLength.GetLength
    LengthParm.ValueateFromString ("20000mm")
...
```

In this step object of type `StrProfilePtLength` is retrieved in `ObjStrProfilePtLength`. Then `RefPoint`, which is created in the step above is assigned to

property. Then it creates reference to the plane DECK.7 and is assigned to the `Direction` property. To set the length it retrieves the `LengthParm` by using m and its value is set to 20000mm. To set the value `ValuateFromString` method is used.

iv. **Updates created member object**

```
...
    ObjPart.UpdateObject oObjMember
End Sub
```

Method `UpdateObject` updates the created member.

8. **Creates a member using a curve**

Call `CreateMemberCrv` method to create member using a curve. `CreateMemberCrv` method takes a reference to the geometrical set as input parameter and it retur output parameter in `ObjMemberCrv`.

```
...
  Dim ObjMemberCrv As SfdMember
  CreateMemberCrv RefGeometricalSet, ObjMemberCrv
...
```

The method `CreateMemberCrv` is detailed as in the below sub steps.

i. **Creates a empty member and sets category, material, profile type and section parameters**

```
Sub CreateMemberCrv(iRefGeometricalSet As Reference, oObjMember As SfdMember)
  CreateMemberAndSetData iRefGeometricalSet, "SldMember", catStrProfileModeCrv, "Steel A42", "WT18x179.5", "catStrTopCenter",
...
```

In this step Uc calls a method `CreateMemberAndSetData` which creates a empty member under given GeometricalSet and sets some of the properties of the the created member in oObjMember. Third parameter of this method is catStrProfileModeCrv. Here catStrProfileModeCrv means profile is created using a

ii. **Creates two points and a line between them. Later this line will be used for member creation**

```
...
    'Get HybridShapeFactory
    Set ObjHybridShapeFactory = ObjPart.HybridShapeFactory
    'Create a point
    Set ObjHybridShapePointCoord = ObjHybridShapeFactory.AddNewPointCoord(5000, 5000, 0)
    Set RefPoint1 = ObjPart.CreateReferenceFromObject(ObjHybridShapePointCoord)
    'Create one more point
    Set ObjHybridShapePointCoord = ObjHybridShapeFactory.AddNewPointCoord(20000, 20000, 0)
    Set RefPoint2 = ObjPart.CreateReferenceFromObject(ObjHybridShapePointCoord)
    'create line between two points
    Set ObjHybridShapeLinePtPt = ObjHybridShapeFactory.AddNewLinePtPt(RefPoint2, RefPoint1)
    Set RefLine = ObjPart.CreateReferenceFromObject(ObjHybridShapeLinePtPt)
...
```

In this step Uc creates retrieves HybridShapeFactory, then it creates a point by calling method `AddNewPointCoord`. After creating point it creates reference stores it in `RefPoint1`. In the same manner it creates another point and stores its reference in `RefPoint2`. Then it creates a line between these two points and in `RefLine`. This line will be used for member creation.

iii. **Retrieves StrProfileCrv and sets Curve**

```
...
    'Get StrProfileCrv object
    Dim ObjStrProfileCrv As StrProfileCrv
    Set ObjStrProfileCrv = oObjMember.StrProfileCrv
    'set the curve
    ObjStrProfileCrv.Curve = RefLine
...
```

In this step object of type `StrProfileCrv` is retrieved in `ObjStrProfileCrv`. Then `RefLine`, which is created in the step above is assigned to the `Curve` pro path of this curve member is created.

iv. **Updates created member object**

```
...
    ObjPart.UpdateObject oObjMember
End Sub
```

Method `UpdateObject` updates the created member.

9. **Creates a member using a surface and two members(curves)**

Call `CreateMemberSurf2Crvs` method to create member using surface and two a curves. `CreateMemberSurf2Crvs` method takes a reference to the geometrical se as input parameters and it returns created member as output parameter in `ObjMemberSurf2Crv`.

```
...
  Dim ObjMemberSurf2Crv As SfdMember
  CreateMemberSurf2Crvs RefGeometricalSet, ObjMemberPtLimit, ObjMemberPtLength, ObjMemberSurf2Crv
...
```

The method `CreateMemberSurf2Crvs` is detailed as in the below sub steps.

i. **Creates a empty member and sets category, material, profile type and section parameters**

```
Sub CreateMemberSurf2Crvs(iRefGeometricalSet As Reference, iFirstMember As SfdMember, iSecondMember As SfdMember, oObjMember
  CreateMemberAndSetData iRefGeometricalSet, "SldMember", catStrProfileModeSurf2Crvs, "Steel A42", "WT18x179.5", "catStrTopCe
...
```

In this step Uc calls a method `CreateMemberAndSetData` which creates a empty member under given GeometricalSet and sets some of the properties of the the created member in oObjMember. Third parameter of this method is catStrProfileModeSurf2Crvs. Here catStrProfileModeSurf2Crvs means profile is cre and two curves.

ii. **Retrieves StrProfileSurf2Crvs and sets a surface and two curves**

```
...
    'Get StrProfileSurf2Crvs object
    Dim ObjStrProfileSurf2Crvs As StrProfileSurf2Crvs
    Set ObjStrProfileSurf2Crvs = oObjMember.StrProfileSurf2Crvs
    'Set Surface
    Set ObjRefSurface = Manager.GetReferencePlane(ObjPart, 1, "DECK.3")
    Set RefSurface = ObjPart.CreateReferenceFromObject(ObjRefSurface)
    ObjStrProfileSurf2Crvs.Surface = RefSurface
    'Set FirstCurve
    Set RefFirstCrv = iFirstMember.GetDelimitedSupport
    ObjStrProfileSurf2Crvs.FirstCurve = RefFirstCrv
    'Set SecondCurve
    Set RefSecondCrv = iSecondMember.GetDelimitedSupport
    ObjStrProfileSurf2Crvs.SecondCurve = RefSecondCrv
...
```

This step sets a surface and two curves. Member will be created on the surface and between two given curves.

Uc first retrieves the object of type `StrProfileSurf2Crvs` in `ObjStrProfileSurf2Crvs`. To set the surface, it first retrieves the reference to the plane DEC the `Surface` property. To set the FirstCurve, it retrieves the delimited support of the `iFirstMember` by using method `GetDelimitedSupport` and assigns it property. In the same manner, it retrieves the delimited support of the `iSecondMember` and assigns it to the `SecondCurve` property.

iii. **Updates created member object**

```
...
    ObjPart.UpdateObject oObjMember
End Sub
```

Method `UpdateObject` updates the created member.

10. **Creates a member using two points**

Call `CreateMemberPtPt` method to create member using two points. `CreateMemberPtPt` method takes a reference to the geometrical set as input parameter and it member as output parameter in `ObjMemberPtPt`.

```
...
  Dim ObjMemberPtPt As SfdMember
  CreateMemberPtPt RefGeometricalSet, ObjMemberPtPt
...
```

The method `CreateMemberPtPt` is detailed as in the below sub steps.

i. **Creates a empty member and sets category, material, profile type and section parameters**

```
Sub CreateMemberPtPt(iRefGeometricalSet As Reference, oObjMember As SfdMember)
  CreateMemberAndSetData iRefGeometricalSet, "SldMember", catStrProfileModePts, "Steel A42", "WT18x179.5", "catStrTopCenter",
...
```

In this step Uc calls a method `CreateMemberAndSetData` which creates a empty member under given GeometricalSet and sets some of the properties of the the created member in oObjMember. Third parameter of this method is catStrProfileModePts. Here catStrProfileModePts means profile is created using two

ii. **Creates two points and and references to these points. Later these points are used for member creation**

```
...
    'Create Start point
    Set ObjHybridShapeFactory = ObjPart.HybridShapeFactory
    Set StartPoint = ObjHybridShapeFactory.AddNewPointCoord(25000, 25000, 0)
    Set RefStartPoint = ObjPart.CreateReferenceFromObject(StartPoint)
    'Create End point
    Set EndPoint = ObjHybridShapeFactory.AddNewPointCoord(25000, 25000, 20000)
    Set RefEndPoint = ObjPart.CreateReferenceFromObject(EndPoint)
...
```

This creates two points and stores its references in `RefStartPoint` and `RefEndPoint`. Member will be created between these two points.

iii. **Retrieves StrProfilePtPt object and sets StartPoint and EndPoint**

```
...
    'Get StrProfilePtPt object
    Dim ObjStrProfilePtPt As StrProfilePtPt
    Set ObjStrProfilePtPt = oObjMember.StrProfilePtPt
    'Set StartPoint
    ObjStrProfilePtPt.StartPoint = RefStartPoint
    'Set EndPoint
    ObjStrProfilePtPt.EndPoint = RefEndPoint
...
```

This step sets a two points. Member will be created in between these two points. Uc first retrieves the object of type `StrProfilePtPt` in `ObjStrProfilePt` StartPoint, it assigns the `RefStartPoint`(reference to the point) to the `StartPoint` property. To set the EndPoint, it assigns the `RefEndPoint`(refernce to th `EndPoint` property.

iv. **Updates created member object**

```
...
    ObjPart.UpdateObject oObjMember
End Sub
```

Method `UpdateObject` updates the created member.

11. **Updates the Part object**

```
...
    ObjPart.Update
End Sub
```

`Update` method updates the `ObjPart`

**Detailed steps of method CreateMemberAndSetData**

- **CreateMemberAndSetData method**

  This method creates a new empty member, sets some properties of the member and returns the created member in `oObjMember` object. Properties which will set by category, profile type, material, secion name and section's anchor point.

```
Sub CreateMemberAndSetData(iRefGeometricalSet As Reference, iCategory As String, iProfileType As CATStrProfileMode, iMaterial As S
                           iSectionName As String, iAnchorPoint As String, oObjMember As SfdMember)

'   1- Retrieves SfdFactory object and creates a empty member
    Dim ObjSfdFactory As SfdFactory
    Set ObjSfdFactory = ObjPart.GetCustomerFactory("SfdFactory")
    Set oObjMember = ObjSfdFactory.AddMember(iRefGeometricalSet)

'   2- Retrieves StrCategoryMngt object and sets category
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = oObjMember.StrCategoryMngt
    ObjStrCategoryMngt.SetCategory iCategory

'   3- Sets profile type
    oObjMember.ProfileType = iProfileType

'   4- Retrieves StrMaterialMngt object and sets material
    Dim ObjMaterialMngt As StrMaterialMngt
    Set ObjMaterialMngt = oObjMember.StrMaterialMngt
    ObjMaterialMngt.SetMaterial iMaterial

'   5- Retrieves StrSectionMngt object and sets section parameters
    Dim ObjStrSectionMngt As StrSectionMngt
    Set ObjStrSectionMngt = oObjMember.StrSectionMngt
    ObjStrSectionMngt.SetSectionName iSectionName
    ObjStrSectionMngt.AnchorPoint = iAnchorPoint

End Sub
```

In the first step of this method `SfdFactory` is retrieved in `ObjSfdFactory` by using method `GetCustomerFactory` from `ObjPart`. Then it calls method `AddMembe` empty member and stores it in `oObjMember` which is a output parameter of this method. Method `AddMember` takes reference to the geometrical set as input parame aggregated under this.

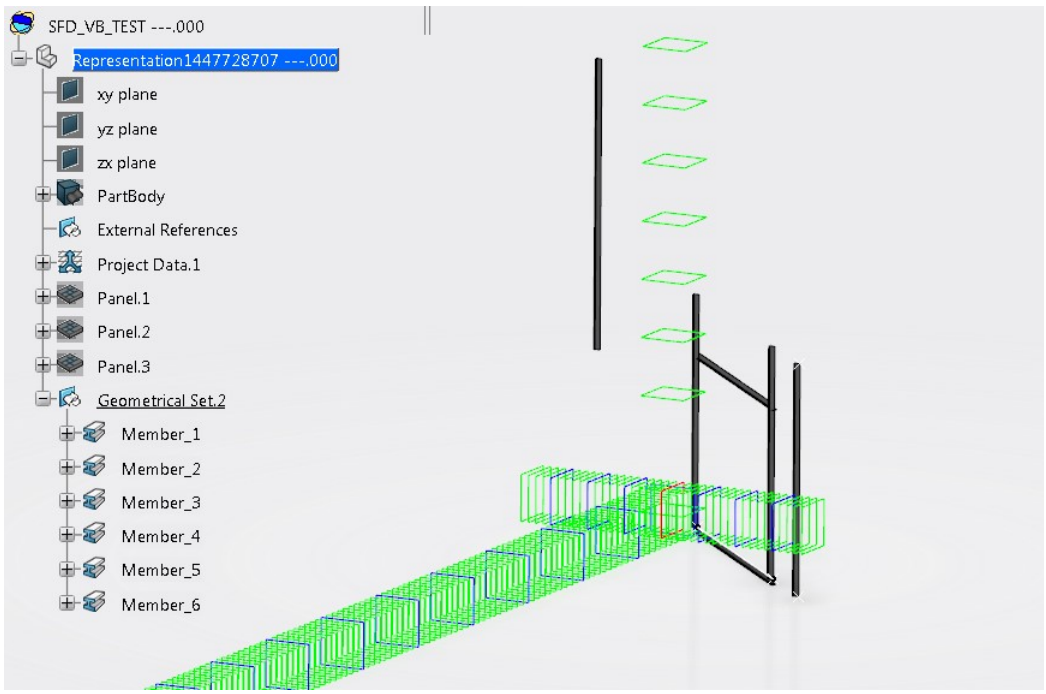
In the second step, it retrieves the object of type `StrCategoryMngt` in `ObjStrCategoryMngt` and calls `SetCategory` method to set the category.

In the third step, it sets `ProfileType` property with `iProfileType`.

In the fourth step, it retrieves the object of type `StrMaterialMngt` in `ObjStrMaterialMngt` and calls `SetMaterial` method to set the material.

In the fifth step, it retrieves the object of type `StrSectionMngt` in `ObjStrSectionMngt` and calls `SetSectionName` method to set the secion name and sets the pro with `iAnchorPoint`.

Fig.1: Member




# Creating an SFD Stiffener

This use case primarily focuses on the methodology to create a SFD stiffener.

Related Topics
Structure Functional Design Object Model Map

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcPart.3dxml`, `CAAScdSfdUcCGR.3dxml` and `CAAScdSfdUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Launching an Automation Use Case

Where to find the macro: CAAScdSfdUcCreateStiffenerSource.htm

This use case can be divided in eight steps:

1. Searches and opens model which is named as "SFD_VB_TEST"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves Service manager
5. Retrieves a SFD panel("panel.1") object
6. Creates a Stiffener
7. Edits created Stiffener
8. Update the Part object

1. **Searches and opens model which is named as "SFD_VB_TEST"**

   As a first step, the UC retrieves a model "SFD_VB_TEST" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_VB_TEST"
 OpenProduct prdName , SFDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor i returned in **`SFDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **`SFDPrdEditor`** is used.

```
 ...
    Set SFDProdSel = SFDPrdEditor.Selection
 ...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves Service Manager**

   In this step UC retrieves object of RfgService.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

   *GetService* method returns the service `RfgService` in `Manager` variable. This service provides various services.

5. **Retrieves a SFD panel ("panel.1") object**

   In this step UC finds a SFD panel in the part and retrieve SfdPanel object using Selection object.

```
...
  Set RefSfdPanel = ObjPart.FindObjectByName("Panel.1")
  Dim ObjSfdPanel As SfdPanel
  SFDProdSel.Add RefSfdPanel
  Set ObjSfdPanel = SFDProdSel.FindObject("CATIASfdPanel")
...
```

   In above lines, *FindObjectByName* method finds object whose name is "Panel.1" and returns reference to it. Here `RefSfdPanel` is of type `Reference`. To retrieve `SfdPanel` object from the reference, add retrieved reference to the selection and call *FindObject* method as shown above. This will give the `SfdPanel` object.

6. **Creates a Stiffener**

   Now panel is available to create stiffener on it. Call `CreateStiffener` method to create stiffener. `CreateStiffener` method takes a panel as input and created sti is returned in output parameter.

```
...
  Dim ObjSfdStiffener As StrSfdStiffener
  CreateStiffener ObjSfdPanel, ObjSfdStiffener
...
```

   The method `CreateStiffener` is detailed as in the below sub steps.

   i. Retrieves the `StrSfdStiffeners` object from panel. Then `AddStiffener` method from object `StrSfdStiffeners` will create a new empty stiffener.

```
Sub CreateStiffener(iObjSfdPanel As SfdPanel, oObjSfdStiffener As StrSfdStiffener)
    'Get StrSfdStiffeners object
    Dim ObjSfdStiffeners As StrSfdStiffeners
    Set ObjSfdStiffeners = iObjSfdPanel.StrSfdStiffeners
    'Add Stiffener
    Set oObjSfdStiffener = ObjSfdStiffeners.AddStiffener
...
```

   `StrSfdStiffeners` object is retrieved in `ObjSfdStiffeners` variable from `iObjSfdPanel` object. On `ObjSfdStiffeners` object `AddStiffener` method is called to create the empty stiffener. Now Uc needs to set the different properties of the stiffener like material, category etc.

ii. Set different properties of the stiffener like material. Retrieve the `StrMaterialMngt` object from `StrSfdStiffener` object and set the material for the stiffe using `SetMaterial` method.

```
...
    'Get StrMaterialMngt object
    Dim ObjMaterialMngt As StrMaterialMngt
    Set ObjMaterialMngt = oObjSfdStiffener.StrMaterialMngt
    'Set material of the stiffener
    ObjMaterialMngt.SetMaterial("Steel A42")
...
```

`ObjMaterialMngt` object is of type `StrMaterialMngt`. It is retrieved from the stiffener `oObjSfdStiffener`. `SetMaterial` method is called to set the mater object `ObjMaterialMngt`.

iii. To set the category retrieve the `StrCategoryMngt` object from `StrSfdStiffener` object and set category using `SetCategory` method.

```
...
    'Get StrCategoryMngt object
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = oObjSfdStiffener.StrCategoryMngt
    'Set category of the stiffener
    ObjStrCategoryMngt.SetCategory("SldStiffener")
...
```

iv. Sets the `ProfileType` property of the stiffener to catStrProfileModeSurfSurf (here catStrProfileModeSurfSurf means profile is created with the intersection surfaces).

```
...
    'Set type of the stiffener
    oObjSfdStiffener.ProfileType = catStrProfileModeSurfSurf
...
```

v. Retrieves the `StrSectionMngt` object from the created stiffener object and sets the different section parameters like section name, anchor point, web orienta flange orientation.

```
...
    'Get StrSectionMngt object
    Dim ObjStrSectionMngt As StrSectionMngt
    Set ObjStrSectionMngt = oObjSfdStiffener.StrSectionMngt
    'Set different section parameters
    ObjStrSectionMngt.SetSectionName ("WT18x179.5")
    ObjStrSectionMngt.AnchorPoint = "catStrTopCenter"
    ObjStrSectionMngt.WebOrientation = 1
    ObjStrSectionMngt.FlangeOrientation = 1
...
```

`ObjStrSectionMngt` object which is of type `StrSectionMngt` is used to set the different section properties of the stiffener. `SetSectionName` method sets th section name for the stiffener. `AnchorPoint` property is used to set anchor point to "catStrTopCenter". Similarly `WebOrientation` and `FlangeOrientatior` properties are set to 1.

vi. Retrieves the `StrProfileLimitMngt` object from the created stiffener object and sets the Start limit and End limit of the stiffener.

```
...
    'Get StrProfileLimitMngt object
    Dim ObjStrProfileLimitMngt As StrProfileLimitMngt
    Set ObjStrProfileLimitMngt = oObjSfdStiffener.StrProfileLimitMngt
    'Set the profile limits
    Set ObjStartLimit = Manager.GetReferencePlane(ObjPart, 2, "CROSS.70")
    Set StartLimit = ObjPart.CreateReferenceFromObject(ObjStartLimit)
    ObjStrProfileLimitMngt.SetLimitingObject 1, StartLimit
    Set ObjEndLimit = Manager.GetReferencePlane(ObjPart, 2, "CROSS.110")
    Set EndLimit = ObjPart.CreateReferenceFromObject(ObjEndLimit)
    ObjStrProfileLimitMngt.SetLimitingObject 2, EndLimit
...
```

`ObjStrProfileLimitMngt` is retrieved from the `oObjSfdStiffener` object. Variable `StartLimit` and `EndLimit` are of type `Reference` which contains references to planes "CROSS.70" and "CROSS.110" respectively. Reference to plane is retrieved using `GetReferencePlane` method on `Manager` object. `Ma` object is of type `RfgService`. `ObjStrProfileLimitMngt` calls `SetLimitingObject` method to set the start and end limit of the of the stiffener. `SetLimitingObject` method hass two input parameters. First input defines whether it is start of the stiffener or end of the stiffener. Second input is the limi object.

vii. Retrieves the `StrProfileSurfSurf` object from the created stiffener object and sets two surfaces which are intersecting with each other. At the intersection these two surfaces stiffener will be created.

```
...
    'Get StrProfileSurfSurf object
    Dim ObjStrProfileSurfSurf As StrProfileSurfSurf
    Set ObjStrProfileSurfSurf = oObjSfdStiffener.StrProfileSurfSurf
    'Set two surfaces
    Dim DMS As Reference
    Set DMS = iObjSfdPanel.GetDelimitedSupport
    ObjStrProfileSurfSurf.FirstSurface = DMS
    Set ObjWebSupport = Manager.GetReferencePlane(ObjPart, 3, "LONG.-4")
    Set WebSupport = ObjPart.CreateReferenceFromObject(ObjWebSupport)
    ObjStrProfileSurfSurf.SecondSurface = WebSupport
...
```

`ObjStrProfileSurfSurf` object is retrieved from `oObjSfdStiffener` object. Here Uc is setting two intersecting surfaces for stiffener creation. To set first surface, it retrieves delimited surface of the `iObjSfdPanel` using method `GetDelimitedSupport` in `DMS` object. Then `DMS` is assigned to the `FirstSurface` property of the `ObjStrProfileSurfSurf` object. Similarly `SecondSurface` is set.

viii. Updates the created stiffener.

```
...
    ObjPart.UpdateObject oObjSfdStiffener
End Sub
...
```

7. **Edits created stiffener**

   In this step UC edits stiffener's some of the data like it changes the support of the stiffener then it changes anchor point, web orientation and flange orientation. Fo editing the stiffener data `EditStiffener` method is called.

```
...
  'Edits the stiffener
  EditStiffener ObjSfdStiffener
...
```

   The method `EditStiffener` is detailed as in the below sub steps.

   i. Retrieves the `StrProfileSurfSurf` object from the stiffener object. Then changes the support of the Stiffener to LONG.0.

```
Sub EditStiffener(iObjSfdStiffener As StrSfdStiffener)
    'Get StrProfileSurfSurf object
    Dim ObjStrProfileSurfSurf As StrProfileSurfSurf
    Set ObjStrProfileSurfSurf = iObjSfdStiffener.StrProfileSurfSurf
    'Change the web support of the stiffener
    Set ObjWebSupport = Manager.GetReferencePlane(ObjPart, 3, "LONG.0")
    Set WebSupport = ObjPart.CreateReferenceFromObject(ObjWebSupport)
    ObjStrProfileSurfSurf.SecondSurface = WebSupport
...
```

   ii. Retrieves the `StrSectionMngt` object from the created stiffener object and changes anchor point, web orientation and flange orientation.

```
...
    'Get StrSectionMngt object
    Dim ObjStrSectionMngt As StrSectionMngt
    Set ObjStrSectionMngt = iObjSfdStiffener.StrSectionMngt
    'Change AnchorPoint, WebOrientation, FlangeOrientation
    ObjStrSectionMngt.AnchorPoint = "catStrTopLeft"
    ObjStrSectionMngt.WebOrientation = -1
    ObjStrSectionMngt.FlangeOrientation = -1
...
```

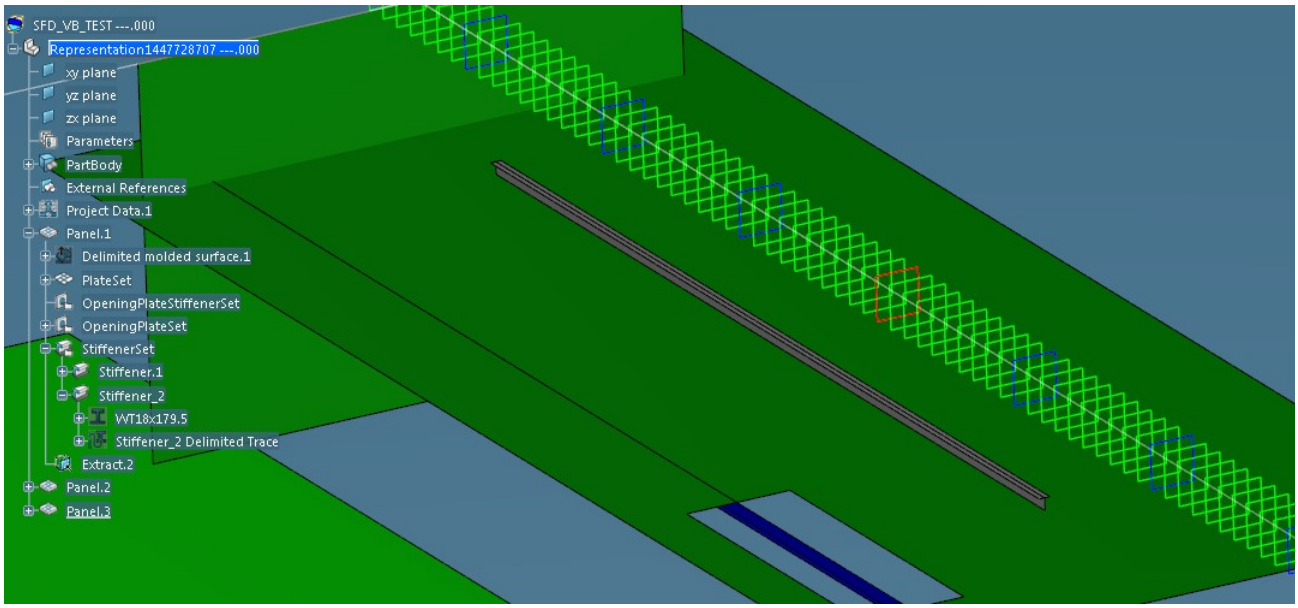
   iii. Updates the stiffener object.

```
...
    'Update the stiffener
    ObjPart.UpdateObject iObjSfdStiffener
End Sub
...
```

8. **Update the `Part` object**

```
...
    ObjPart.Update
End Sub
```

Fig.1: Stiffener with Limits




# Creating an SFD StiffenerOnFreeEdge on Panel Limits

This use case primarily focuses on the methodology to create a SFD StiffenerOnFreeEdge(SFE) on panel limits.

Related Topics
Structure Functional Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcPart.3dxml`, `CAAScdSfdUcCGR.3dxml` and `CAAScdSfdUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: CAAScdSfdUcCreateSFEOnLimitsSource.htm

This use case can be divided in seven steps:

1. Searches and opens model which is named as "SFD_VB_TEST"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves a SFD panel object
5. Creates SFE on panel limits
6. Edits SFE
7. Updates the Part object

1. **Searches and opens model which is named as "SFD_VB_TEST"**

   As a first step, the UC retrieves a model "SFD_VB_TEST" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_VB_TEST"
 OpenProduct prdName , SFDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor i returned in **`SFDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **`SFDPrdEditor`** is used.

```
 ...
    Set SFDProdSel = SFDPrdEditor.Selection
 ...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves a SFD panel object**

   In this step UC retrieves panel object.

```
...
  Set RefSfdPanel = ObjPart.FindObjectByName("Panel.1")
  Dim ObjSfdPanel As SfdPanel
  SFDProdSel.Add (RefSfdPanel)
  Set ObjSfdPanel = SFDProdSel.FindObject("CATIASfdPanel")
...
```

   `FindObjectByName` method finds object whose name is "Panel.1" and returns reference to it. Here RefSfdPanel is of type `Reference`. To retrieve `SfdPanel` objec `Reference` add retrieved reference to the selection and call `FindObject` method as shown above. This will return the `SfdPanel` object variable.

5. **Creates SFE on panel limits**

   Now, panel is available to create SFE on its limits. Call `CreateSFE` method to create SFE on panel limits. `CreateSFE` method takes a panel as input parameter and created SFE as output parameter in `ObjSfdStiffenerOnFreeEdge`.

```
...
  Dim ObjSfdStiffenerOnFreeEdge As StrSfdStiffenerOnFreeEdge
  CreateSFE ObjSfdPanel, ObjSfdStiffenerOnFreeEdge
...
```

   The method `CreateSFE` is detailed as in the below sub steps.

   i. **Retrieves `StrSfdStiffeners` and creates a empty SFE**

```
Sub CreateSFE(iObjSfdPanel As SfdPanel, oObjSfdStiffenerOnFreeEdge As StrSfdStiffenerOnFreeEdge)
    'Get StrSfdStiffeners object
    Dim ObjSfdStiffeners As StrSfdStiffeners
    Set ObjSfdStiffeners = iObjSfdPanel.StrSfdStiffeners
    'Create StiffenerOnFreeEdge
    Set oObjSfdStiffenerOnFreeEdge = ObjSfdStiffeners.AddStiffenerOnFreeEdge
...
```

   `StrSfdStiffeners` is retrieved in `ObjSfdStiffeners` variable from `iObjSfdPanel` object. On `StrSfdStiffeners` object, `AddStiffenerOnFreeEdge` me called to create the SFE with no data. Empty SFE is created now Uc sets different properties of the SFE like material, section etc.

   ii. **Retrieves `StrCategoryMngt` and sets category**

```
...
    'Get StrCategoryMngt object
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = oObjSfdStiffenerOnFreeEdge.StrCategoryMngt
    'set category
    ObjStrCategoryMngt.SetCategory "SldStiffenerOnFreeEdge"
    'set automatic naming to true
    ObjStrCategoryMngt.AutomaticName = True
...
```

   Object of `StrCategoryMngt` is retrieved in `ObjStrCategoryMngt`. Then method `SetCategory` is called to set the category. The property `AutomaticName` is enable the automatic naming of objects.

   iii. **Retrieves `StrMaterialMngt` and sets material**

```
...
    'Get StrMaterialMngt object
    Dim ObjMaterialMngt As StrMaterialMngt
    Set ObjMaterialMngt = oObjSfdStiffenerOnFreeEdge.StrMaterialMngt
    'set material
    ObjMaterialMngt.SetMaterial ("Steel A42")
...
```

Object of `StrMaterialMngt` is retrieved in `ObjMaterialMngt`. Then method `SetMaterial` is called to set the material.

iv. **Sets section parameters**

Call `SetSectionParameters` method to set the section parameters. This method takes 3 input parameters SFE object, section name and anchor point.

```
...
  SetSectionParameters oObjSfdStiffenerOnFreeEdge, "WT18x179.5", "catStrTopCenter"
...
```

The method `SetSectionParameters` is detailed as in the below sub steps.

i. **Retrieves `StrSectionMngt`**

```
Sub SetSectionParameters(iObjSfdStiffenerOnFreeEdge As StrSfdStiffenerOnFreeEdge, iStrSecName As String, iAnchorPoint As
    Dim ObjStrSectionMngt As StrSectionMngt
    Set ObjStrSectionMngt = iObjSfdStiffenerOnFreeEdge.StrSectionMngt
...
```

Object of type `StrSectionMngt` is retrieved in `ObjStrSectionMngt` from `iObjSfdStiffenerOnFreeEdge` object.

ii. **Sets section parameters**

```
...
    ObjStrSectionMngt.SetSectionName (iStrSecName)
    ObjStrSectionMngt.AnchorPoint = iAnchorPoint
    ObjStrSectionMngt.WebOrientation = 1
    ObjStrSectionMngt.InvertWebOrientation
    ObjStrSectionMngt.FlangeOrientation = 1
    ObjStrSectionMngt.InvertFlangeOrientation
    'set angle
    Dim AngleParm As Parameter
    Set AngleParm = ObjStrSectionMngt.GetAngle
    AngleParm.ValuateFromString ("60deg")
End Sub
```

Method `SetSectionName` sets the section name. Anchor point is set using `AnchorPoint` property. `WebOrientation` is set to 1 and method `InvertWebOrientation` is called to invert the web orientation. Same is done for FlangeOrientation property. User can directly set WebOrienation an FlangeOrientation to -1. But to introduce you to the methods `InvertWebOrientation` & `InvertFlangeOrientation` Uc uses it here. To set the Web the SFE, the parameter for WebAngle is retrieved using method `GetAngle` and then its value is set using method `ValuateFromString`.

v. **Sets profile type**

```
...
    'Set profile type (catStrProfileModeOnLimits = profile on limits)
    oObjSfdStiffenerOnFreeEdge.ProfileType = catStrProfileModeOnLimits
...
```

Type of the profile is set using property `ProfileType`. This property is set to catStrProfileModeOnLimits. catStrProfileModeOnLimits means that the creat will be on edges of the panel.

vi. **Retrieves `StrProfileOnLimits` and sets limits**

```
...
    'Get StrProfileOnLimits object
    Dim ObjStrProfileOnLimits As StrProfileOnLimits
    Set ObjStrProfileOnLimits = oObjSfdStiffenerOnFreeEdge.StrProfileOnLimits
    'Store indices of plate limits on which SFE should be created in an array
    Dim LimitIndexList(1) As Variant
    LimitIndexList(0) = 1
    LimitIndexList(1) = 4
    'Set limits
    ObjStrProfileOnLimits.SetLimits (LimitIndexList)
...
```

Object of `StrProfileOnLimits` is retrieved in `ObjStrProfileOnLimits` from object `oObjSfdStiffenerOnFreeEdge`. Then limit indices of panel on whic should be created are stored in an array `LimitIndexList`. Method `SetLimits` is called to set the limits. This method takes one input parameter i.e. limits ar

vii. **Updates created SFE object**

```
...
    ObjPart.UpdateObject oObjSfdStiffenerOnFreeEdge
End Sub
```

Method `UpdateObject` updates the created SFE.

6. **Edits SFE**

Method `EditStiffenerOnFreeEdge` is called to edit the created SFE. `EditStiffenerOnFreeEdge` method takes created SFE as input parameter. It will be edited

```
...
  EditStiffenerOnFreeEdge ObjSfdStiffenerOnFreeEdge
...
```

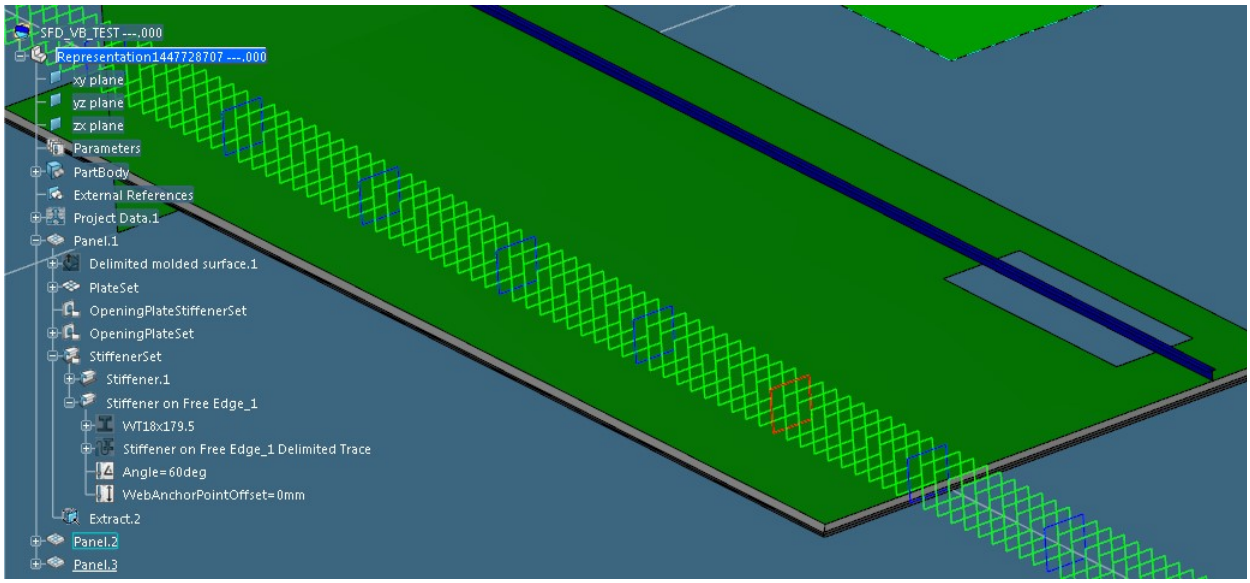
The method `EditStiffenerOnFreeEdge` first retrieves the object of type `StrProfileOnLimits`. Then it creates an array LimitIndexList of size 2, whose contents {2,4}. This means SFE's previous support limits are changed to 2 and 4. So SFE will be created at panel limit 2 and panel limit 4. Then it calls method `ObjStrProfileOnLimits.SetLimits` and updates the SFE object. Go to the CAAScdSfdUcCreateSFEOnLimitsSource.htm to check the source of this method.

7. **Updates the Part object**

```
...
    ObjPart.Update
End Sub
```

`Update` method updates the `ObjPart`

Fig.1: StiffenerOnFreeEdge on Panel Limits




# Creating an SFD StiffenerOnFreeEdge on an Opening

This use case primarily focuses on the methodology to create a SFD StiffenerOnFreeEdge(SFE) on an opening's edge.

Related Topics

Structure Functional Design Object Model Map

Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcPart.3dxml`, `CAAScdSfdUcCGR.3dxml` and `CAAScdSfdUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: CAAScdSfdUcCreateSFEOnOpeningSource.htm

This use case can be divided in seven steps:

1. Searches and opens model which is named as "SFD_VB_TEST"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves a SFD panel object
5. Retrieves a SFD opening object
6. Creates SFE on opening
7. Updates the Part object

1. **Searches and opens model which is named as "SFD_VB_TEST"**

   As a first step, the UC retrieves a model "SFD_VB_TEST" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_VB_TEST"
 OpenProduct prdName , SFDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor is returned in **`SFDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **`SFDPrdEditor`** is used.

```
 ...
    Set SFDProdSel = SFDPrdEditor.Selection
 ...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves a SFD panel object**

In this step UC retrieves panel object.

```
...
  Set RefObjPanel = ObjPart.FindObjectByName("Panel.1")
  Dim ObjPanel As SfdPanel
  SFDProdSel.Add (RefObjPanel)
  Set ObjPanel = SFDProdSel.FindObject("CATIASfdPanel")
...
```

`FindObjectByName` method finds object whose name is "Panel.1" and returns reference to it. Here RefObjPanel is of type `Reference`. To retrieve `SfdPanel` object from `Reference` add retrieved reference to the selection and call `FindObject` method as shown above. This will return the `SfdPanel` object variable.

5. **Retrieves a SFD opening object**

In this step UC retrieves opening object. SFE will be created on this opening's edge.

```
...
  Set RefObjStrOpening = ObjPart.FindObjectByName("OpeningPlate.2")
  Dim ObjStrOpening As StrOpening
  SFDProdSel.Add (RefObjStrOpening)
  Set ObjStrOpening = SFDProdSel.FindObject("CATIAStrOpening")
...
```

`FindObjectByName` method finds object whose name is "OpeningPlate.2" and returns reference to it. Here RefObjStrOpening is of type `Reference`. To retrieve `StrOpening` object from `Reference` add retrieved reference to the selection and call `FindObject` method as shown above. This will return the `StrOpening` object variable.

6. **Creates SFE on opening**

Now, panel and opening are available to create SFE on opening edge. Call `CreateSFEOnOpening` method to create SFE on opening. `CreateSFEOnOpening` method takes a panel and an opening as input parameters and it returns created SFE as output parameter in `ObjSFE`.

```
...
  Dim ObjSFE As StrSfdStiffenerOnFreeEdge
  CreateSFEOnOpening ObjPanel, ObjStrOpening, ObjSFE
...
```

The method `CreateSFEOnOpening` is detailed as in the below sub steps.

i. **Retrieves `StrSfdStiffeners` and creates a empty SFE.**

```
Sub CreateSFEOnOpening(iObjSfdPanel As SfdPanel, iObjStrOpening As StrOpening, oObjSFE As StrSfdStiffenerOnFreeEdge)
    'Get StrSfdStiffeners object
    Dim ObjSfdStiffeners As StrSfdStiffeners
    Set ObjSfdStiffeners = iObjSfdPanel.StrSfdStiffeners
    'Create StiffenerOnFreeEdge
    Set oObjSFE = ObjSfdStiffeners.AddStiffenerOnFreeEdge
...
```

`StrSfdStiffeners` is retrieved in `ObjSfdStiffeners` variable from `iObjSfdPanel` object. On `StrSfdStiffeners` object, `AddStiffenerOnFreeEdge` method is called to create the SFE with no data. Empty SFE is created now Uc sets different properties of the SFE like material, section etc.

ii. **Retrieves `StrMaterialMngt` and sets material**

```
...
    'Get StrMaterialMngt object
    Dim ObjStrMaterialMngt As StrMaterialMngt
    Set ObjStrMaterialMngt = oObjSFE.StrMaterialMngt
    'Set material
    ObjStrMaterialMngt.SetMaterial ("Steel A42")
...
```

Here Uc retrieves object of type `StrMaterialMngt` in `ObjStrMaterialMngt`. Then `SetMaterial` method is called on object `ObjStrMaterialMngt`. Method `SetMaterial` takes material name as input parameter.

iii. **Sets profile type**

```
...
    'set profile type
    oObjSFE.ProfileType = catStrProfileModeOnOpening
...
```

Here Uc sets `ProfileType` property of `SFE`. When a profile is created on an opening's edge then profile type needs to be set to catStrProfileModeOnOpening.

iv. **Retrieves StrSectionMngt and sets section parameters**

```
...
    'Get StrSectionMngt object
    Dim ObjStrSectionMngt As StrSectionMngt
    Set ObjStrSectionMngt = oObjSFE.StrSectionMngt
    'Set different section parameters
    ObjStrSectionMngt.SetSectionName ("WT18x179.5")
    ObjStrSectionMngt.AnchorPoint = "catStrTopCenter"
    ObjStrSectionMngt.WebOrientation = 1
    ObjStrSectionMngt.FlangeOrientation = 1
...
```

Here Uc retrieves object of type `StrSectionMngt` in `ObjStrSectionMngt`. Then using `SetSectionName` method section name is set to WT18x179.5. `AnchorPoint` property is set to "catStrTopCenter". Properties `WebOrientation` and `FlangeOrientation` are set to 1.

v. **Retrieves `StrProfileOnOpening` and sets opening, on which SFE will be created**

```
...
    'Get StrProfileOnOpening object
    Dim ObjStrProfileOnOpening As StrProfileOnOpening
    Set ObjStrProfileOnOpening = oObjSFE.StrProfileOnOpening
```

```
    'set opening on which SFE will be created
    ObjStrProfileOnOpening.Opening = iObjStrOpening
...
```

Here Uc retrieves object of type `StrProfileOnOpening` in `ObjStrProfileOnOpening`. Then on `ObjStrProfileOnOpening` object `Opening` property is set. This property defines that SFE will be created on this opening's edge. /p>

vi. **Updates created SFE object**

```
...
    ObjPart.UpdateObject oObjSFE
End Sub
```

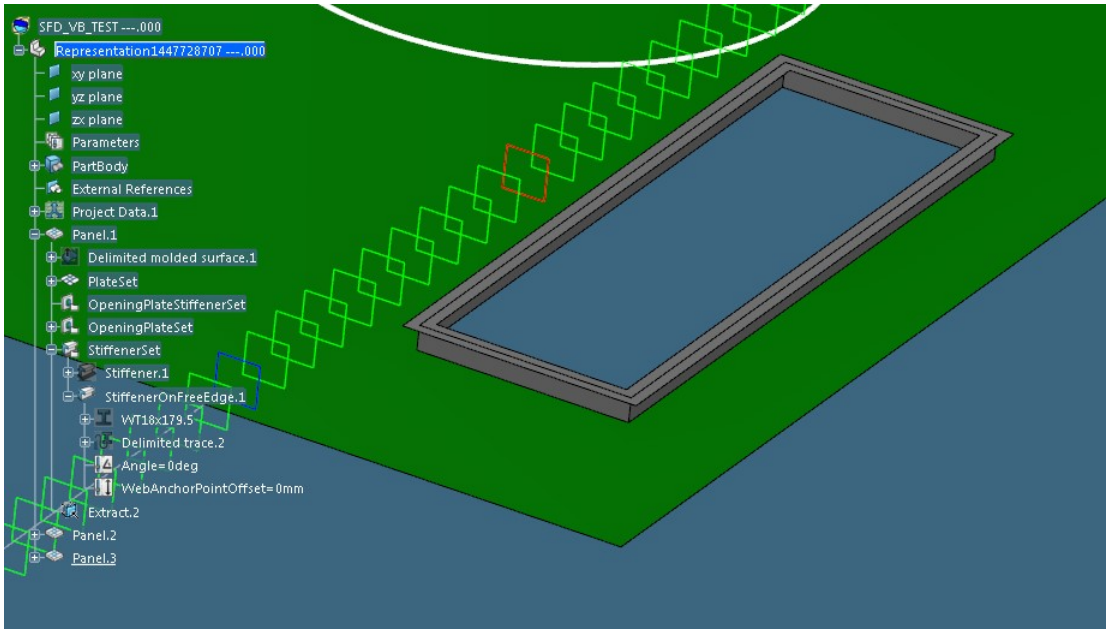
`UpdateObject` method from `Part` will update the created SFE.

7. **Updates the Part object**

```
...
    ObjPart.Update
End Sub
```

`Update` method updates the `ObjPart`

Fig.1: StiffenerOnFreeEdge on Opening




# Creating an Endcut

This use case primarily focuses on the methodology to create a Endcut.

Related Topics
Structure Functional Design Object Model Map
Launching an Automation Use Case

---

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcCGR.3dxml` `CAAScdSfdUcSR.3dxml` `CAAScdSfdUcPart.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: CAAScdSfdUcCreateEndcutSource.htm

This use case can be divided in eight steps:

1. Searches and opens model which is named as "SFD_VB_TEST"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves a SFD stiffener object
5. Creates Endcut
6. Edits Endcut
7. Retrieves StrEndcutMngt object and remove the Endcut created at the start of the profile
8. Updates the Part object

1. **Searches and opens model which is named as "SFD_VB_TEST"**

As a first step, the UC retrieves a model "SFD_VB_TEST" from DB and loads it and returns object of the Editor.

```
...
  Dim SFDPrdEditor  As  Editor
  Dim prdName As String
  prdName = "SFD_VB_TEST"
  OpenProduct prdName , SFDPrdEditor
...
```

The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor is returned in **`SFDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **SFDPrdEditor** is used.

```
...
  Set SFDProdSel = SFDPrdEditor.Selection
...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves a SFD Stiffener object**

   In this step UC retrieves stiffener object. Endcut will be created on this stiffener.

```
...
  Set RefObjSfdStiffener = ObjPart.FindObjectByName("Stiffener.1")
  Dim ObjSfdStiffener As StrSfdStiffener
  SFDProdSel.Add (RefObjSfdStiffener)
  Set ObjSfdStiffener = SFDProdSel.FindObject("CATIASfdStiffener")
...
```

   `FindObjectByName` method finds object whose name is "Stiffener.1" and returns reference to it. Here RefObjSfdStiffener is of type `Reference`. To retrieve `StrSfdStiffener` object from `Reference` add retrieved reference to the selection and call `FindObject` method as shown above. This will return the `StrSfdStiffener` object variable.

5. **Creates Endcut**

   Now, stiffener i.e. profile is available to create Endcut. Call `CreateEndCut` method to create Endcut. `CreateEndCut` method takes a stiffener as input parameter and it returns created Endcut as output parameter.

```
...
  Dim ObjStrEndCut As StrEndCut
  CreateEndCut ObjSfdStiffener, ObjStrEndCut
...
```

   The method `CreateEndCut` is detailed as in the below sub steps.

   i. **Retrieves `StrEndcutMngt` object and and create an Endcut with no data.**

```
Sub CreateEndCut(iObjSfdStiffener As StrSfdStiffener, oObjStrEndCut As StrEndCut)
'Get StrEndcutMngt object
  Dim ObjStrEndcutMngt As StrEndcutMngt
  Set ObjStrEndcutMngt = iObjSfdStiffener.StrEndcutMngt
'Create Endcut
  Set oObjStrEndCut = ObjStrEndcutMngt.AddEndCut(1)
...
```

   `StrEndcutMngt` is retrieved in `ObjStrEndcutMngt` variable from `iObjSfdStiffener` object. On `ObjStrEndcutMngt` object, `AddEndCut` method is called to create the Endcut with no data. Method `AddEndCut` takes one input parameter which defines whether this Endcut is created at the start of the profile or end of the profile. 1 is for start and 2 is for end. Now Uc needs to set the different properties of the Endcut like Endcut's parameters etc.

   ii. **Sets Endcut parameters**

```
...
    SetEndCutParameters oObjStrEndCut, iObjSfdStiffener
...
```

   Method `SetEndCutParameters` is called to set the Endcut parameters.

   The method `SetEndCutParameters` is detailed as in the below sub steps.

   i. **Retrieves `StrDetailFeature` object**

```
Sub SetEndCutParameters(iObjStrEndCut As StrEndCut,iObjSfdStiffener As StrSfdStiffener)
    Dim ObjStrDetailFeature As StrDetailFeature
    Set ObjStrDetailFeature = iObjStrEndCut.StrDetailFeature
...
```

   `ObjStrDetailFeature` is of type `StrDetailFeature`. It is retrieved from `iObjStrEndCut` object using method `StrDetailFeature`. Later this object is used to set different Endcut parameters like Endcut name, Endcut type etc.

   ii. **Sets Endcut type**

```
...
    ObjStrDetailFeature.Type = "SNIPE"
...
```

   `Type` property of `StrDetailFeature` is set to "SNIPE". This defines the type of the Endcut.

   iii. **Sets Endcut name**

```
...
    ObjStrDetailFeature.FeatureName = "WT_snipe_radius"
...
```

   `FeatureName` property of `StrDetailFeature` is set to "WT_snipe_radius". This defines the name of the Endcut.

   iv. **Retrieves list of parameters of impacted object**

```
...
    Set ObjPartParameters = ObjPart.Parameters
  'parameters will be aggregated under impacted
  'object(here impacted object is iObjSfdStiffener)
    Set HBParameters = ObjPartParameters.SubList(iObjSfdStiffener, True)
...
```

`Parameters` property of `ObjPart` returns collection object containing the part parameters. `SubList` method from `Parameters` returns the sub-collection of parameters aggregated to `iObjSfdStiffener` object and it collected in `HBParameters` object.

v. **Creates parameters and stores role of each parameter in an array**

```
...
    Set R1 = HBParameters.CreateDimension("R1", "LENGTH", "50")
  'Define parameter roles in an array
    Dim ParamRoles(0) As Variant
    ParamRoles(0) = "R1"
...
```

Method `CreateDimension` creates a user dimension. Here Uc creates one parameter R1 for Endcut. Role of this parameter is stored in ParamRoles array.

vi. **Sets parameters**

```
...
    ObjStrDetailFeature.SetParameters HBParameters, ParamRoles
...
```

`SetParameters` method of `ObjStrDetailFeature` is called to set the parameters for Endcut creation. This method takes two input parameters. First parameter is the collection object which contains Endcut parameters and second object is an array which contains the role of each parameter from the collection object.

vii. **Updates the ObjStrDetailFeature object**

```
...
    ObjStrDetailFeature.Update
End Sub
```

Call to `Update` method of `ObjStrDetailFeature` will update connection coordinate and visualization of the Endcut feature.

iii. **Retrieves `SfdConnectionSet` object from SfdPart object and updates the ConnectionSet**

```
...
    Dim ObjSfdPart As SfdPart
    SFDProdSel.Add (ObjPart)
    Set ObjSfdPart = SFDProdSel.FindObject("CATIASfdPart")
    Dim ObjSfdConnectionSet As SfdConnectionSet
    Set ObjSfdConnectionSet = ObjSfdPart.GetConnectionsSet
    ObjSfdConnectionSet.UpdateConnectionsSet UpdatedCnx, RemovedCnx, UnkStatusCnx
...
```

To retrieve `ObjSfdPart` from `ObjPart`, ObjPart is added to the `Selection` object `SFDProdSel`. Then on `SFDProdSel` object, `FindObject` method is called as shown above. `GetConnectionSet` method from `SfdPart` returns `SfdConnectionSet` which is collected in `ObjSfdConnectionSet`. Then `UpdateConnectionsSet` updates all the connections created. It has three out parameters, first is for number of updated connections second is for number of connections which are removed and third is for number of unknown status connections.

iv. **Updates created Endcut object**

```
...
    ObjPart.UpdateObject oObjStrEndCut
End Sub
```

`UpdateObject` method from `Part` will update the created Endcut.

6. **Edits Endcut**

In this step UC edits Endcut. It changes the SNIPE Endcut to Weld Endcut.

```
...
   EditEndcut ObjStrEndCut
...
```

The method `EditEndcut` is detailed as in the below sub steps.

i. **Retrieves StrDetailFeature object using created Endcut**

```
Sub EditEndcut(iObjStrEndCut As StrEndCut)
Dim ObjStrDetailFeature As StrDetailFeature
Set ObjStrDetailFeature = iObjStrEndCut.StrDetailFeature
...
```

Method `StrDetailFeature` retrieves the object of type `StrDetailFeature`.

ii. **Changes the Endcut Type and Name**

```
...
    ObjStrDetailFeature.Type = "Weld"
    ObjStrDetailFeature.FeatureName = "Metal To Metal"
...
```

Uc changes the Endcut type from "SNIPE" to "Weld" and name from "WT_snipe_radius" to "Metal To Metal". Here no need to set the parameters because no parameter is exposed to the user to change for this type of Endcut.

iii. **Updates the connection coordinate and visualiztion for Endcut**

```
...
    ObjStrDetailFeature.Update
End Sub
```

Call to `Update` method of `ObjStrDetailFeature` will update connection coordinate and visualization of the Endcut feature.

7. **Retrieves StrEndcutMngt object and remove the Endcut created at the start of the profile**

   In this step UC removes Endcut created on stiffener.

```
...
    'Get StrEndcutMngt object
    Set ObjStrEndcutMngt = ObjSfdStiffener.StrEndcutMngt
    'Remove the Endcut
    ObjStrEndcutMngt.RemoveEndcut (1)
...
```

   The method `RemoveEndcut` is called to remove the created Endcut. It takes one input parameter which defines whether this Endcut is at the start of the profile or at the end of the profile. 1 defines start of the profile and 2 defines end of the profile.

8. **Updates the Part object**

```
...
  ObjPart.Update
End Sub
```

   `Update` method updates the `ObjPart`

# Creating a SFD Slot

This use case primarily focuses on the methodology to create a slot.

Related Topics
Structure Functional Design Object Model Map
Launching an Automation Use Case

---

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcSteel_A42.3dxml`, `CAAScdSfdUcWT18x179_5.3dxml`, `CAAScdSfdUcSlotSections.3dxml` `CAAScdSfdUcCreateSlot.3dxml`, `CAAScdSfdUcCGR.3dxml` and `CAAScdSfdUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: CAAScdSfdUcCreateSlotSource.htm

This use case can be divided in nine steps:

1. Searches and opens model which is named as "SFD_VB_TEST"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves a SFD stiffener object
5. Retrieves a SFD panel object
6. Creates Slot
7. Edits Slot data
8. Removes created slot from the panel
9. Updates the Part object

1. **Searches and opens model which is named as "SFD_VB_TEST"**

   As a first step, the UC retrieves a model "SFD_VB_TEST" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_VB_TEST"
 OpenProduct prdName , SFDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor is returned in **`SFDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **`SFDPrdEditor`** is used.

```
...
    Set SFDProdSel = SFDPrdEditor.Selection
...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves a SFD Stiffener object**

   In this step UC create and retrieves stiffener object. This stiffener will be used as a penetrating element for the slot creation.

```
...
  Dim ObjSfdStiffener As StrSfdStiffener
  CreateStiffener ObjSfdStiffener
...
```

`CreateStiffener` method will create a stiffener and return it in `ObjSfdStiffener` object.

5. **Retrieves a SFD panel object**

   In this step UC finds a SFD panel in the part and retrieve SfdPanel object using Selection object. This panel will be used as a penetrated element in the slot creation.

```
...
  Set RefObjSfdPanel = ObjPart.FindObjectByName("Panel.3")
  Dim ObjSfdPanel As SfdPanel
  SFDProdSel.Add RefObjSfdPanel
  Set ObjSfdPanel = SFDProdSel.FindObject("CATIASfdPanel")
...
```

   In above lines, `FindObjectByName` method finds object whose name is "Panel.3" and returns reference to it. Here `RefObjSfdPanel` is of type `Reference`. To retrieve `SfdPanel` object from the reference, add retrieved reference to the selection and call `FindObject` method as shown above. This will give the `SfdPanel` object.

6. **Creates Slot**

   Now, penetrated element (panel) and penetrating element (stiffener) are available to create slot. Call `CreateSlot` method to create slot. `CreateSlot` method takes a panel and a stiffener as input parameters and it returns created slot as output parameter.

```
...
  Dim ObjStrSlot As StrSlot
  CreateSlot ObjSfdPanel, ObjSfdStiffener, ObjStrSlot
...
```

   The method `CreateSlot` is detailed as in the below sub steps.

   i. **Retrieves `StrSlots` object and and create a slot with no data.**

```
Sub CreateSlot(iObjSfdPanel As SfdPanel, iObjSfdStiffener As StrSfdStiffener, oObjStrSlot As StrSlot)
    'Get StrSlots object
    Dim ObjStrSlots As StrSlots
    Set ObjStrSlots = iObjSfdPanel.StrSlots
    'Create StrSlot
    Set oObjStrSlot = ObjStrSlots.Add
...
```

   `StrSlots` is retrieved in `ObjStrSlots` variable from `iObjSfdPanel` object. On `ObjStrSlots` object, `Add` method is called to create the slot with no data. Now Uc needs to set the different properties of the slot like penetrating element, slot parameters etc.

   ii. **Sets penetrating profile for slot**

```
...
    'set penetrating element
    Dim PenetratingElem As Reference
    Set PenetratingElem = ObjPart.CreateReferenceFromObject(iObjSfdStiffener)
    oObjStrSlot.SetPenetratingProfile PenetratingElem
...
```

   `PenetratingElem` is of type `Reference`. Method `CreateReferenceFromObject` creates reference to `iObjSfdStiffener` and returns it in `PenetratingElem`. Then `SetPenetratingProfile` method is called to set the penetrating profile for the slot.

   iii. **Sets slot parameters**

```
...
    SetSlotParameters oObjStrSlot, iObjSfdPanel
...
```

   Method `SetSlotParameters` is called to set the slot parameters of respective slot which is set.

   The method `SetSlotParameters` is detailed as in the below sub steps.

   i. **Retrieves `StrDetailFeature` object**

```
Sub SetSlotParameters(iObjStrSlot As StrSlot,iObjSfdPanel As SfdPanel)
    Dim ObjStrDetailFeature As StrDetailFeature
    Set ObjStrDetailFeature = iObjStrSlot.StrDetailFeature
...
```

   `ObjStrDetailFeature` is of type `StrDetailFeature`. It is retrieved from `iObjStrSlot` object using method `StrDetailFeature`. Later this object is used to set different slot parameters like slot name, slot type etc.

   ii. **Sets Slot type**

```
...
    ObjStrDetailFeature.Type = "RECT"
...
```

   `Type` property of `StrDetailFeature` is set to "RECT". This defines the type of the slot.

   iii. **Sets Slot name**

```
...
    ObjStrDetailFeature.FeatureName = "RECT_UNI_WT"
...
```

   `FeatureName` property of `StrDetailFeature` is set to "RECT_UNI_WT". This defines the name of the slot.

   iv. **Retrieves list of parameters of impacted object**

```
...
    Set ObjPartParameters = ObjPart.Parameters
```

```
    'parameters will be aggregated under impacted
    'object(here impacted object is iObjSfdPanel)
    Set HBParameters = ObjPartParameters.SubList(iObjSfdPanel, True)
...
```

`Parameters` property of `ObjPart` returns collection object containing the part parameters. `SubList` method from `Parameters` returns the sub-collection of parameters aggregated to `iObjSfdPanel` object and it collected in `HBParameters` object.

v. **Creates parameters and stores role of each parameter in an array**

```
...
    Set DBB = HBParameters.CreateDimension("DBB", "LENGTH", "300")
    Set DR = HBParameters.CreateDimension("DR", "LENGTH", "200")
    Set DL = HBParameters.CreateDimension("DL", "LENGTH", "200")
    Set DTT = HBParameters.CreateDimension("DTT", "LENGTH", "50")
    'Define parameter roles in an array
    Dim ParamRoles(3) As Variant
    ParamRoles(0) = "DBB"
    ParamRoles(1) = "DR"
    ParamRoles(2) = "DL"
    ParamRoles(3) = "DTT"
...
```

Method `CreateDimension` creates a user dimension. Here Uc creates 4 parameters (DBB, DR, DL, DLL) for slot. Role of each parameter is stored in ParamRoles array.

vi. **Sets parameters**

```
...
    ObjStrDetailFeature.SetParameters HBParameters, ParamRoles
...
```

`SetParameters` method of `ObjStrDetailFeature` is called to set the parameters for slot creation. This method takes two input parameters. First parameter is the collection object of slot parameters and second object is an array which contains the role of each parameter from the collection object.

vii. **Updates the ObjStrDetailFeature object**

```
...
    ObjStrDetailFeature.Update
End Sub
```

Call to `Update` method of `ObjStrDetailFeature` will update connection coordinate and visualization of the slot feature.

iv. **Retrieves `SfdConnectionSet` object from SfdPart object and updates the ConnectionSet**

```
...
    Dim ObjSfdPart As SfdPart
    SFDProdSel.Add (ObjPart)
    Set ObjSfdPart = SFDProdSel.FindObject("CATIASfdPart")
    Dim ObjSfdConnectionSet As SfdConnectionSet
    Set ObjSfdConnectionSet = ObjSfdPart.GetConnectionsSet
    ObjSfdConnectionSet.UpdateConnectionsSet UpdatedCnx, RemovedCnx, UnkStatusCnx
...
```

To retrieve `ObjSfdPart` from `ObjPart`, ObjPart is added to the `Selection` object `SFDProdSel`. Then on `SFDProdSel` object, `FindObject` method is called as shown above. `GetConnectionSet` method from `SfdPart` returns `SfdConnectionSet` which is collected in `ObjSfdConnectionSet`. Then `UpdateConnectionsSet` updates all the connections created. It has three out parameters, first is for number of updated connections second is for number of connections which are removed and third is for number of unknown status connections.

v. **Updates created Slot object**

```
...
    ObjPart.UpdateObject oObjStrSlot
End Sub
```

`UpdateObject` method from `Part` will update the created slot.

7. **Edits Slot data**

In this step UC edits Slot's some of the data like it changes the value of the first parameter which is DBB.

```
...
   EditSlotData ObjStrSlot
...
```

The method `EditSlotData` is detailed as in the below sub steps.

i. **Retrieves StrDetailFeature object using created slot**

```
Sub EditSlotData(iObjStrSlot As StrSlot)
    Dim ObjStrDetailFeature As StrDetailFeature
    Set ObjStrDetailFeature = iObjStrSlot.StrDetailFeature
...
```

Property `StrDetailFeature` retrieves the object of type `StrDetailFeature`.

ii. **Retrieves slot parameters**

```
...
    Dim SlotParameters As StrParameters
    Set SlotParameters  = ObjStrDetailFeature.GetParameters
...
```

Method `GetParameters` retrieves the collection object of type `StrParameters`. This object has slot parameters which were created and set by the method `SetParameters`.

iii. **Changes the slot parameter value**

```
...
    Dim DBBParameter As Parameter
    Set DBBParameter = SlotParameters.Item(1)
    DBBParameter.ValuateFromString("500mm")
...
```

Uc retrieves first parameter from the collection of parameters using method `Item(1)`. Here 1 represents the first parameter from the collection.

iv. **Updates the connection coordinate and visualiztion for slot**

```
...
    ObjStrDetailFeature.Update
End Sub
```

Call to `Update` method of `ObjStrDetailFeature` will update connection coordinate and visualization of the slot feature.

8. **Removes created slot from the panel**

In this step UC removes slot created on a panel.

```
...
   RemoveSlot ObjSfdPanel
...
```

The method `RemoveSlot` is detailed as in the below sub steps.

i. **Retrieves object of type StrSlots**

```
Sub RemoveSlot (iObjSfdPanel As SfdPanel)
    Dim ObjStrSlots As StrSlots
    Set ObjStrSlots = iObjSfdPanel.StrSlots
...
```

`StrSlots` provides methods for creating a new slot retrieve the created slots and remove slots.

ii. **Removes the first slot**

```
...
    Dim ObjStrSlot As StrSlot
    Set ObjStrSlot = ObjStrSlots.Item(1)
    ObjStrSlots.Remove (ObjStrSlot)
End Sub
```

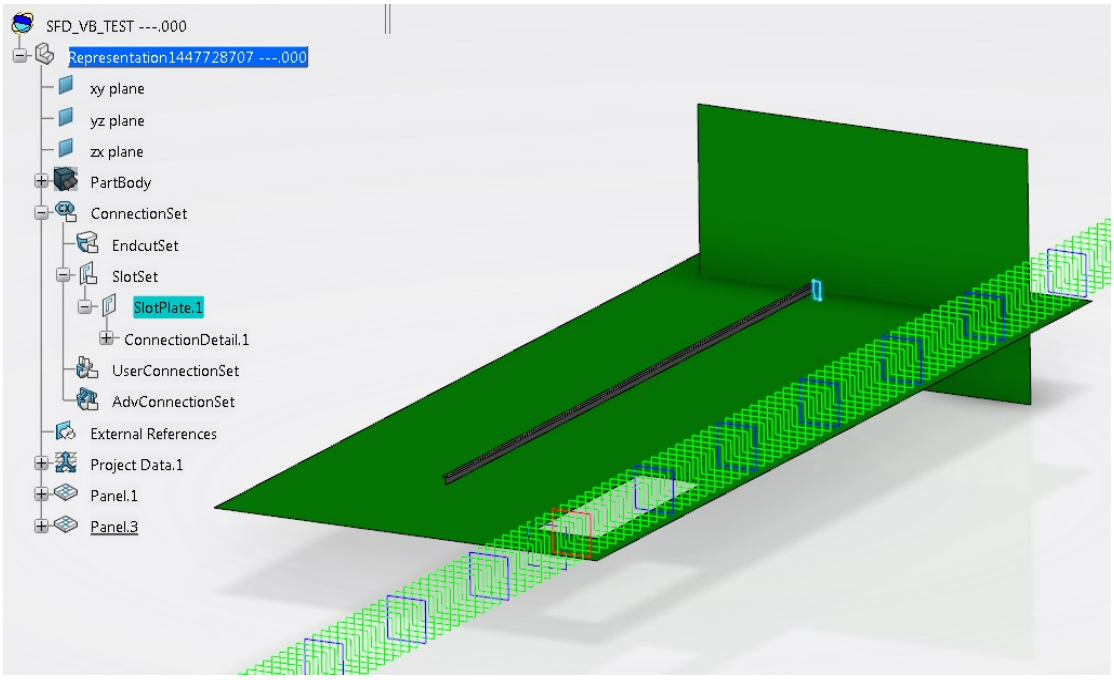
Call to the `Item(1)` retrieves first slot from the collection of slot. Then **RemoveSlot** method from `StrSlots` removes the slot `ObjStrSlot`. `RemoveSlot` method takes one input parameter which is a object of `StrSlot`. This is the slot which will be removed.

9. **Updates the Part object**

```
...
    ObjPart.Update
End Sub
```

`Update` method updates the `ObjPart`.

Fig.1: Slot




# Creating an SFD Opening Using 3D Objects

This use case primarily focuses on the methodology to create a SFD Opening using 3D object.

Related Topics
Structure Functional Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcPart.3dxml` `CAAScdSfdUcCGR.3dxml` and `CAAScdSfdUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: CAAScdSfdUcCreateOpening3DObjectSource.htm

This use case can be divided in seven steps:

1. Searches and opens model which is named as "SFD_VB_TEST"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves a SFD panel object
5. Creates opening
6. Removes opening
7. Updates the Part object

1. **Searches and opens model which is named as "SFD_VB_TEST"**

   As a first step, the UC retrieves a model "SFD_VB_TEST" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_VB_TEST"
 OpenProduct prdName , SFDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor is returned in **`SFDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **`SFDPrdEditor`** is used.

```
 ...
    Set SFDProdSel = SFDPrdEditor.Selection
 ...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves a SFD panel object**

   In this step UC retrieves panel object.

```
...
  Set RefObjPanel = ObjPart.FindObjectByName("Panel.1")
  Dim ObjSfdPanel As SfdPanel
  SFDProdSel.Add RefObjPanel
  Set ObjSfdPanel = SFDProdSel.FindObject("CATIASfdPanel")
...
```

   `FindObjectByName` method finds object whose name is "Panel.1" and returns reference to it. Here RefObjPanel is of type `Reference`. To retrieve `SfdPanel` object from `Reference` add retrieved reference to the selection and call `FindObject` method as shown above. This will return the `SfdPanel` object variable.

5. **Creates opening**

   Now, panel is available to create opening on it. Call `CreateOpening3DObject` method to create opening on panel. `CreateOpening3DObject` method takes a panel as input parameter and it returns created opening as output parameter in `ObjStrOpening`.

```
...
  Dim ObjStrOpening As StrOpening
  CreateOpening3DObject ObjSfdPanel, ObjStrOpening
...
```

   The method `CreateOpening3DObject` is detailed as in the below sub steps.

   i. **Creates a cylinder, later which will be used for opening creation as a intersecting profile**

```
Sub CreateOpening3DObject(iObjSfdPanel As SfdPanel, oObjStrOpening As StrOpening)
    Set ObjHybridShapeFactory = ObjPart.HybridShapeFactory
    Set ObjHybridShapePointCoord = ObjHybridShapeFactory.AddNewPointCoord(90000, 10000, 0)
    Set PointRef = ObjPart.CreateReferenceFromObject(ObjHybridShapePointCoord)
    Set Manager = CATIA.ActiveEditor.GetService("RfgService")
    Set ObjDirection = Manager.GetReferencePlane(ObjPart, 1, "DECK.2")
    Set Direction = ObjPart.CreateReferenceFromObject(ObjDirection)
    Set ObjHybridShapeDirection = ObjHybridShapeFactory.AddNewDirection(Direction)
    Set ObjHybridShapeCylinder = ObjHybridShapeFactory.AddNewCylinder(PointRef, 2000, 32800, 20, ObjHybridShapeDirection)
    Set CylinderRef = ObjPart.CreateReferenceFromObject(ObjHybridShapeCylinder)
...
```

   In this step Uc creates a cylinder and stores its reference in `CylinderRef`. Later this created cylinder will be used for opening creation as a intersecting element.

   ii. **Retrieves `StrOpenings` and creates an opening with no properties set**

```
...
    'Get StrOpenings object
    Dim ObjStrOpenings As StrOpenings
    Set ObjStrOpenings = iObjSfdPanel.GetOpenings(0)
    'Add opening
    Set oObjStrOpening = ObjStrOpenings.Add
...
```

Object of `StrOpenings` is retrieved in `ObjStrOpenings`. Then method `Add` from `ObjStrOpenings`is called to create an opening. This creates an opening with no properties set. Now Uc sets different properties on opening to complete the creation of the opening.

iii. **Sets opening type to 3d object**

```
...
    ObjStrOpeningType.OpeningType = catStrOpeningMode3DObject
...
```

`OpeningType` property is set to catStrOpeningMode3DObject. catStrOpeningMode3DObject defines that opening is created using a 3D object.

iv. **Retrieves `StrCategoryMngt` and sets category**

```
...
    'Get StrCategoryMngt object
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = oObjStrOpening.StrCategoryMngt
    'set category
    ObjStrCategoryMngt.SetCategory "SldOpening"
...
```

Object of `StrCategoryMngt` is retrieved in `ObjStrCategoryMngt`. Then method `SetCategory` is called to set the category.

v. **Retrieves `StrOpening3DObject` and sets intersecting profile**

```
...
    'Get StrOpening3DObject object
    Dim ObjStrOpening3DObject As StrOpening3DObject
    Set ObjStrOpening3DObject = oObjStrOpening.StrOpening3DObject
    'Set intersecting element
    ObjStrOpening3DObject.IntersectingElement = CylinderRef
...
```

Object of `StrOpening3DObject` is retrieved in `ObjStrOpening3DObject`. Then created `CylinderRef` is assigned to the property `IntersectingElement`. Opening will be created at the intersection of panel and cylinder (3D Object).

vi. **Retrieves StrOpeningExtrusionMngt and sets forming extrusion mode**

```
...
    Set ObjStrOpeningExtrusionMngt = oObjStrOpening.StrOpeningExtrusionMngt
    ObjStrOpeningExtrusionMngt.ExtrusionMode = 2
...
```

In this step object forming mode is set. To set the forming mode, forming mode value is assigned to property `ExtrusionMode`. Here 2 defines After Forming Extrusion Mode.

vii. **Retrieves `SfdOpeningPlate` and call method MoveToOpeningPPrSet**

```
...
    'Get SfdOpeningPlate object
    Dim ObjSfdOpeningPlate As SfdOpeningPlate
    Set ObjSfdOpeningPlate = oObjStrOpening.SfdOpeningPlate
    'move opening to OpeningPPrSet so that it will interrupt the profile
    ObjSfdOpeningPlate.MoveToOpeningPPrSet
...
```

Object of `SfdOpeningPlate` is retrieved in `ObjSfdOpeningPlate` from object `oObjSfdOpening`. Then on `ObjSfdOpeningPlate` object `MoveToOpeningPPrSet` method is called. Due to this method call opening will interrupt the profiles if they are intersecting the opening.

viii. **Updates created opening object**

```
...
    ObjPart.UpdateObject oObjStrOpening
End Sub
```

Method `UpdateObject` updates the created opening.

6. **Removes opening**

```
...
    Dim ObjStrOpenings As StrOpenings
    Set ObjStrOpenings = ObjSfdPanel.GetOpenings(0)
    ObjStrOpenings.Remove ObjStrOpening
...
```

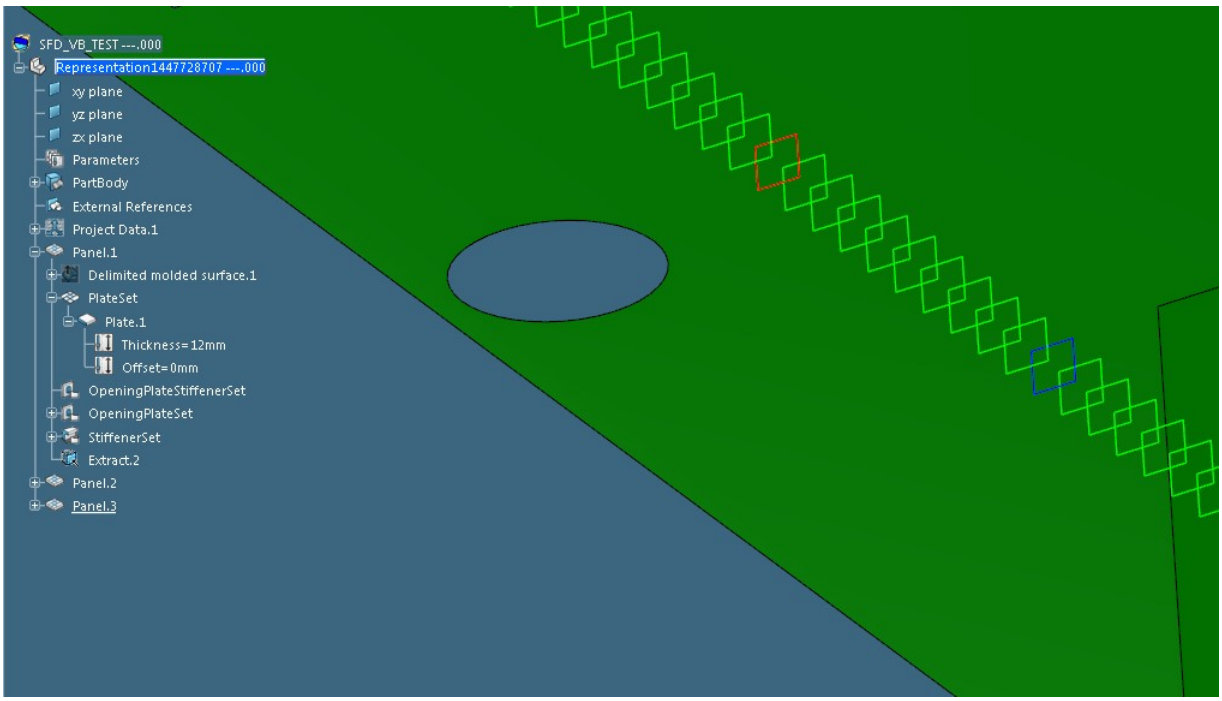
Object of `StrOpenings` is retrieved in `ObjStrOpenings`. Then on `ObjStrOpenings` object `Remove` method is called to remove the opening. This method takes a input parameter opening object to remove.

7. **Updates the Part object**

```
...
    ObjPart.Update
End Sub
```

`Update` method updates the `ObjPart`

Fig.1: Opening using 3D object (before remove Opening)




# Creating an SFD Opening Using a Sketch Profile

This use case primarily focuses on the methodology to create a SFD Opening using sketch profile.

Related Topics
Structure Functional Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcPart.3dxml`, `CAAScdSfdUcCGR.3dxml` and `CAAScdSfdUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: CAAScdSfdUcCreateOpeningSketchSource.htm

This use case can be divided in nine steps:

1. Searches and opens model which is named as "SFD_VB_TEST"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves a SFD panel object
5. Retrieves first sketch profile
6. Creates opening with limit mode set to UpToLast
7. Retrieves second sketch profile
8. Creates opening with limit mode set to dimensions(Limit1/Limit2)
9. Updates the Part object

1. **Searches and opens model which is named as "SFD_VB_TEST"**

   As a first step, the UC retrieves a model "SFD_VB_TEST" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_VB_TEST"
 OpenProduct prdName , SFDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor is returned in **`SFDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **`SFDPrdEditor`** is used.

```
 ...
    Set SFDProdSel = SFDPrdEditor.Selection
 ...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves a SFD panel object**

   In this step UC retrieves panel object.

```
...
  Set RefSfdPanel = ObjPart.FindObjectByName("Panel.1")
  Dim ObjSfdPanel As SfdPanel
  SFDProdSel.Add RefSfdPanel
  Set ObjSfdPanel = SFDProdSel.FindObject("CATIASfdPanel")
...
```

`FindObjectByName` method finds object whose name is "Panel.1" and returns reference to it. Here RefSfdPanel is of type `Reference`. To retrieve `SfdPanel` object from `Reference` add retrieved reference to the selection and call `FindObject` method as shown above. This will return the `SfdPanel` object variable.

5. **Retrieves first sketch profile**

   In this step UC retrieves a sketch profile.

```
...
  Set ProfileSketchOpeningUpToLast = ObjPart.FindObjectByName("Profile.1")
  Set RefProfileSketchOpeningUpToLast = ObjPart.CreateReferenceFromObject(ProfileSketchOpeningUpToLast)
...
```

`FindObjectByName` method finds object whose name is "Profile.1" and returns reference to it. Here RefProfileSketchOpeningUpToLast is of type `Reference`. To retrieve the `RefProfileSketchOpeningUpToLast` object from `ProfileSketchOpeningUpToLast` object `CreateReferenceFromObject` method is used.

6. **Creates opening with limit mode set to UpToLast**

   Now, panel and sketch profile are available to create opening. Call `CreateOpeningSketchUpToLast` method to create opening on panel using sketch profile. `CreateOpeningSketchUpToLast` method takes a panel and a sketch profile as input parameters and it returns created opening as output parameter in `ObjStrOpeningWtLtMdUpToLast`.

```
...
  Dim ObjStrOpeningWtLtMdUpToLast As StrOpening
  CreateOpeningSketchUpToLast ObjSfdPanel, RefProfileSketchOpeningUpToLast, ObjStrOpeningWtLtMdUpToLast
...
```

The method `CreateOpeningSketchUpToLast` is detailed as in the below sub steps.

   i. **Creates a opening with no properties set**

```
Sub CreateOpeningSketchUpToLast(iObjSfdPanel As SfdPanel, iRefProfileSketch As Reference, oObjStrOpening As StrOpening)
  AddOpening iObjSfdPanel, oObjStrOpening
...
```

   In this step Uc calls a method `AddOpening` which creates a opening with no properties set and returns it in output parameter `oObjStrOpening`. Uc sets different properties like catgory, type of opening, etc. of the opening in the subsequent steps.

   ii. **Sets opening type to catStrOpeningModeOutputProfile for sketch opening**

```
...
  SetOpeningType oObjStrOpening, catStrOpeningModeOutputProfile
...
```

   In this step Uc calls a method `SetOpeningType`. This method takes a opening (oObjStrOpening) and opening type value (catStrOpeningModeOutputProfile) as input parameters.

   iii. **Sets category**

```
...
  SetCategory oObjStrOpening, "SldOpening"
...
```

   In this step Uc calls a method `SetCategory`, it sets the category of the opening. This method takes a opening (oObjStrOpening) and category ("SldOpening") as input parameters.

   iv. **Retrieves StrOpeningExtrusionMngt and sets forming extrusion mode**

```
...
    Set ObjStrOpeningExtrusionMngt = oObjStrOpening.StrOpeningExtrusionMngt
    ObjStrOpeningExtrusionMngt.ExtrusionMode = 2
...
```

   In this step object forming extrusion mode is set. To set the forming extrusion mode, extrusion mode value is assigned to property `ExtrusionMode`. Here 2 defines After Forming Extrusion Mode.

   v. **Sets output profile**

```
...
  'set LimitMode to 0 for UpTOLast
  SetOutPutProfile oObjStrOpening, iRefProfileSketch, 0
...
```

   In this step Uc calls a method `SetOutPutProfile`. It sets the output profile, direction and limit mode for the opening. This method takes a opening (oObjStrOpening), reference to sketch profile(iRefProfileSketch) and limit mode value(0) as input parameters. Here 0 means limit mode(Extrusion) is set to "UpToLast".

   vi. **Updates created opening object**

```
...
    ObjPart.UpdateObject oObjStrOpening
End Sub
```

   Method `UpdateObject` updates the created opening.

7. **Retrieves second sketch profile**

```
...
  Set ProfileSketchOpeningDimensions = ObjPart.FindObjectByName("Profile.2")
```

```
  Set RefProfileSketchOpeningDimensions = ObjPart.CreateReferenceFromObject(ProfileSketchOpeningDimensions)
...
```

`FindObjectByName` method finds object whose name is "Profile.2" and returns reference to it. Here `RefProfileSketchOpeningDimensions` is of type `Reference`. To retrieve the `RefProfileSketchOpeningDimensions` object from `ProfileSketchOpeningDimensions` object `CreateReferenceFromObject` method is used.

8. **Creates opening with limit mode set to dimensions(Limit1/Limit2)**

Call `CreateOpeningSketchDimensions` method to create opening on panel using sketch profile. `CreateOpeningSketchDimensions` method takes a panel and a sketch profile as input parameters and it returns created opening as output parameter in `ObjStrOpeningWtLtMdDim`.

```
...
  Dim ObjStrOpeningWtLtMdDim As StrOpening
  CreateOpeningSketchDimensions ObjSfdPanel, RefProfileSketchOpeningDimensions, ObjStrOpeningWtLtMdDim
...
```

The method `CreateOpeningSketchDimensions` is detailed as in the below sub steps.

i. **Creates a opening with no properties set**

```
Sub CreateOpeningSketchDimensions(iObjSfdPanel As SfdPanel, iRefProfileSketch As Reference, oObjStrOpening As StrOpening)
  AddOpening iObjSfdPanel, oObjStrOpening
...
```

In this step Uc calls a method `AddOpening` which creates a opening with no properties set and returns it in output parameter `oObjStrOpening`. Uc sets different properties like catgory, type of opening, etc. of the opening in the subsequent steps.

ii. **Sets opening type to 1 for sketch opening**

```
...
  SetOpeningType oObjStrOpening, catStrOpeningModeOutputProfile
...
```

In this step Uc calls a method `SetOpeningType`. This method takes a opening (oObjStrOpening) and opening type value (catStrOpeningModeOutputProfile) as input parameters. Here catStrOpeningModeOutputProfile is means the opening is sketch opening.

iii. **Sets category**

```
...
  SetCategory oObjStrOpening, "SldOpening"
...
```

In this step Uc calls a method `SetCategory`, it sets the category of the opening. This method takes a opening (oObjStrOpening) and category ("SldOpening") as input parameters.

iv. **Retrieves StrOpeningExtrusionMngt and sets forming extrusion mode**

```
...
    Set ObjStrOpeningExtrusionMngt = oObjStrOpening.StrOpeningExtrusionMngt
    ObjStrOpeningExtrusionMngt.ExtrusionMode = 2
...
```

In this step object forming mode is set. To set the forming mode, forming mode value is assigned to property `ExtrusionMode`. Here 2 defines After Forming Extrusion Mode.

v. **Sets output profile**

```
...
  'set LimitMode to 1 for Dimensions
  SetOutPutProfile oObjStrOpening, iRefProfileSketch, 1
...
```

In this step Uc calls a method `SetOutPutProfile`. It sets the output profile, direction and limit mode for the opening. This method takes a opening (oObjStrOpening), reference to sketch profile(iRefProfileSketch) and limit mode value(1) as input parameters. Here 1 means limit mode(Extrusion) is set to "Limit1/Limit2".

vi. **Retrieves StrOpeningLimitDimensionsMngt object**

```
...
  Dim ObjStrOpeningLimitDimensionsMngt As StrOpeningLimitDimensionsMngt
  Set ObjStrOpeningLimitDimensionsMngt = oObjStrOpening.StrOpeningLimitDimensionsMngt
...
```

Uc retrieves object of type `StrOpeningLimitDimensionsMngt` in `ObjStrOpeningLimitDimensionsMngt`. This is used to set the dimensions.

vii. **Retrieves and sets FirstOffset and SecondOffset parameter values of opening and invert it**

```
...
  'set FirstOffset of opening output profile
  Set FirstOffset = ObjStrOpeningLimitDimensionsMngt.GetFirstOffset
  FirstOffset.ValuateFromString ("1000mm")
  'Set second Offset of opening output profile
  Set SecondOffset = ObjStrOpeningLimitDimensionsMngt.GetSecondOffset
  SecondOffset.ValuateFromString ("-2000mm")
  'Invert the first offset and second offset
  ObjStrOpeningLimitDimensionsMngt.Invert
...
```

Here Uc retrieves `FirstOffset` parameter by using method `GetFirstOffset` and its value is set to 1000mm by using method `ValuateFromString`. Then Uc retrieves `SecondOffset` parameter and its value is set to -2000mm by using method `ValuateFromString`. These offsets are inverted using method `Invert`.

viii. **Updates created opening object**

```
    ...
        ObjPart.UpdateObject oObjStrOpening
    End Sub
```

Method `UpdateObject` updates the created opening.

9. **Updates the Part object**

```
...
    ObjPart.Update
End Sub
```

`Update` method updates the `ObjPart`

**Detailed steps of methods called in the use case**

- **AddOpening method**

```
Sub AddOpening(iObjSfdPanel As SfdPanel, oObjStrOpening As StrOpening)
    'Get StrOpenings object
    Dim ObjStrOpenings As StrOpenings
    Set ObjStrOpenings = iObjSfdPanel.StrOpenings
    'Add opening
    Set oObjStrOpening = ObjStrOpenings.Add
End Sub
```

Method `AddOpening`takes a panel object `iObjSfdpanel` as input parameter and it returns created opening as output parameter in `oObjStrOpening`. In this method object of type `StrOpenings` is retrieved in `ObjStrOpenings`. Then on this object `Add` method is called to create an opening with no properties set.

- **SetOpeningType method**

```
Sub SetOpeningType(iObjStrOpening As StrOpening, iOpeningType As CATStrOpeningMode))
    'set opening type
    iObjStrOpening.OpeningType = iOpeningType
End Sub
```

Method `SetOpeningType` takes a opening object `iObjStrOpening` and opening type `iOpeningType` as input parameters. In this method `iOpeningType` which is input to this method is assigned to `OpeningType` property.

- **SetCategory method**

```
Sub SetCategory(iObjStrOpening As StrOpening, iCategory As String)
    'Get StrCategoryMngt object
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = iObjStrOpening.StrCategoryMngt
    'set catgory
    ObjStrCategoryMngt.SetCategory iCategory
End Sub
```

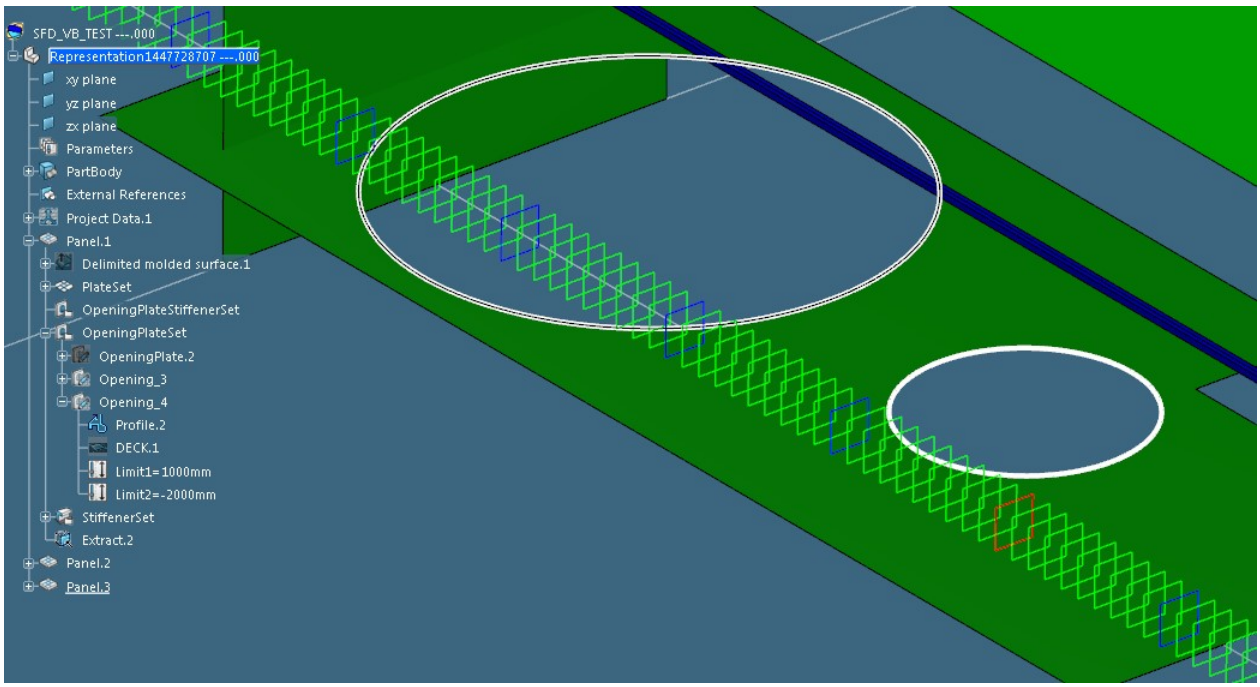
Method `SetCategory` takes opening object(iObjStrOpening) and category(iCategory) as input parameters. In this method object of type `StrCategoryMngt` is retrieved in `ObjStrCategoryMngt`. Then method `SetCategory` is called to set the category.

- **SetOutPutProfile method**

```
Sub SetOutPutProfile(iObjStrOpening As StrOpening, iOutputProfile As Reference, iLimitMode As Long)
'   1- Retrieves StrOpeningOutputProfile
    Dim ObjStrOpeningOutputProfile As StrOpeningOutputProfile
    Set ObjStrOpeningOutputProfile = iObjStrOpening.StrOpeningOutputProfile
'   2- Sets OutputProfile for sketch opening
    ObjStrOpeningOutputProfile.OutputProfile = iOutputProfile
'   3- Sets direction
    Set Manager = CATIA.ActiveEditor.GetService("RfgService")
    Set ObjRefDirection = Manager.GetReferencePlane(ObjPart, 1, "DECK.1")
    Set RefDirection = ObjPart.CreateReferenceFromObject(ObjRefDirection)
    ObjStrOpeningOutputProfile.Direction = RefDirection
'   4- Sets LimitMode
    ObjStrOpeningOutputProfile.LimitMode = iLimitMode
End Sub
```

Method `SetOutPutProfile` takes opening object(iObjStrOpening), output profile(iOutputProfile) and limit mode(iLimitMode) as input parameters. In the first step of this method object of type `StrOpeningOutputProfile` is retrieved in `ObjStrOpeningOutputProfile`. In the second step `OutputProfile` property is set to `iOutputProfile`. In the third step reference to the plane "DECK.1" is retrieved using `GetReferencePlane` method from `RfgService` and then it set as the Direction. In the fourth step it sets `LimitMode` property to iLimitMode.

Fig.1: Sketch Opening




# Creating a Standard Opening

This use case primarily focuses on the methodology to create Standard Opening.

Related Topics
Structure Functional Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcPart.3dxml`, `CAAScdSfdUcCGR.3dxml` and `CAAScdSfdUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: CAAScdSfdUcCreateOpeningStandardSource.htm

This use case can be divided in six steps:

1. Searches and opens model which is named as "SFD_VB_TEST"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves service of type RfgService
5. Retrieves a SFD panel object
6. Create standard Opening using Offset/Offset positioning strategy

1. **Searches and opens model which is named as "SFD_VB_TEST"**

   As a first step, the UC retrieves a model "SFD_VB_TEST" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_VB_TEST"
 OpenProduct prdName , SFDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor i
   **`SFDPrdEditor`**.

2. **Retrieves Selection object**

   In this step UC retrieves Selection object in `SFDProdSel` variable.

```
...
  Set SFDProdSel = SFDPrdEditor.Selection
...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object in `ObjPart` variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves service of type RfgService**

   In this step UC retrieves `RfgService`.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

`GetService` method returns `RfgService`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

5. **Retrieves a SFD panel object**

   In this step UC retrieves panel object.

```
...
  Set RefObjPanel = ObjPart.FindObjectByName("Panel.1")
  Dim ObjSfdPanel As SfdPanel
  SFDProdSel.Add RefObjPanel
  Set ObjSfdPanel = SFDProdSel.FindObject("CATIASfdPanel")
...
```

   `FindObjectByName` method finds object whose name is "Panel.1" and returns reference to it. Here RefObjPanel is of type `Reference`. To retrieve `SfdPanel` objec retrieved reference to the selection and call `FindObject` method as shown above. This will return the `SfdPanel` object variable.

6. **Create Standard Opening with Offset/Offset positioning strategy**

   Call `CreateStandardOpeningOffsetOffset` method to create a standard opening with offset/offset positioning strategy. `CreateStandardOpeningOffsetOffse` panel object as input parameter and returns created opening as output parameter in `ObjStrOpeningOffsetOffset`.

```
...
  Dim ObjStrOpeningOffsetOffset As StrOpening
  CreateStandardOpeningOffsetOffset ObjSfdPanel, ObjStrOpeningOffsetOffset
  ObjPart.Update
...
```

   The method `CreateStandardOpeningOffsetOffset` is detailed as in the below sub steps.

   i. **Retrieves all contour names and selects rectangle contour name**

```
Sub CreateStandardOpeningOffsetOffset(iObjSfdPanel As SfdPanel, oObjStrOpening As StrOpening)
    Dim ContourNames() As Variant
    GetContourNames iObjSfdPanel, ContourNames
    Dim ContourName As String
    ContourName = ContourNames(1)
...
```

      In this step Uc retrieves all the contour names available, by calling method `GetContourNames`. This method takes a panel as input parameter and returns the output parameter. Then Rectangle contour name which is at index 1 is stored in `ContourName` variable.

      The method `GetContourNames` is detailed as in the below sub steps.

      i.
```
Sub GetContourNames(iObjSfdPanel As SfdPanel, oContourNames() As Variant)
    'Get StrOpeningsMgr object
    Dim ObjStrOpeningsMgr As StrOpeningsMgr
    Set ObjStrOpeningsMgr = iObjSfdPanel.StrOpeningsMgr
    'Get available contour names
    ObjStrOpeningsMgr.GetAvailableStandardContours oContourNames
End Sub
```

         In this method object `ObjStrOpeningsMgr` of type `StrOpeningsMgr` is retrieved from iObjSfdPanel. Then method `GetAvailableStandardContours` contour names.

   ii. **Retrieves contour params**

```
...
    Dim StdContourParms As StrStandardContourParameters
    GetContourParams iObjSfdPanel, ContourName, StdContourParms
...
```

      In this step Uc retrieves the parameters of a particular contour. To do this Uc calls a method `GetContourParams`. This method takes a panel object and nam input parameter and it returns the collection of contour parameters as output parameter in `StdContourParms`. Here `StdContourParms` is a collection object `StrStandardContourParameters`, which is a collection of `StrParameter`.

      The method `GetContourParams` is detailed as in the below sub steps.

      i.
```
Sub GetContourParams(iObjSfdPanel As SfdPanel, iContourName As String, oContourParms As StrStandardContourParameters)
    'Get StrOpeningsMgr object
    Dim ObjStrOpeningsMgr As StrOpeningsMgr
    Set ObjStrOpeningsMgr = iObjSfdPanel.StrOpeningsMgr

    'Get parameters of contour
    Set oContourParms = ObjStrOpeningsMgr.GetStandardContourParms(iContourName)
    If oContourParms.Count = 0 Then
        Err.Raise 1, Err.Source, "No contour params found"
        Exit Sub
    End If
End Sub
```

         In this method object `ObjStrOpeningsMgr` of type `StrOpeningsMgr` is retrieved from iObjSfdPanel. Then method `GetStandardContourParms` is cal parameters. Method `GetStandardContourParms` takes contour name as input parameter. Contour parameters related to this name will be returned in

   iii. **Sets contour parameter's data**

```
...
    SetRectContourParamsData StdContourParms, "1000mm", "400mm", "50mm"
...
```

      In this step Uc sets the data of contour parameters by calling method `SetRectContourParamsData`. Method `SetRectContourParamsData` takes contour pa object, width, height and corner radius as input parameters.

   iv. **Retrieves all standard positiong strategy names and selects offset offset positioning strategy**

```
...
    Dim StdPosStrategyNames() As Variant
```

```
    GetStdPosStrategyNames iObjSfdPanel, StdPosStrategyNames
    'Select Positioning strategy offset offset
    Dim PosStratName As String
    PosStratName = StdPosStrategyNames(0)
...
```

In this step Uc retrieves all the standard positioning strategy names available, by calling method `GetStdPosStrategyNames`. This method takes a panel as ir returns the names in an array as output parameter. Then "Offset/Offset" contour name which is at index 0 is stored in `PosStratName` variable.

The method `GetStdPosStrategyNames` is detailed as in the below sub steps.

i.
```
Sub GetStdPosStrategyNames(iObjSfdPanel As SfdPanel, oStdPosStrategyNames() As Variant)
    'Get StrOpeningsMgr object
    Dim ObjStrOpeningsMgr As StrOpeningsMgr
    Set ObjStrOpeningsMgr = iObjSfdPanel.StrOpeningsMgr
    'Get available standard positioning strategy names
    ObjStrOpeningsMgr.GetAvailableStandardPositioningStrategies oStdPosStrategyNames
End Sub
```

In this method object `ObjStrOpeningsMgr` of type `StrOpeningsMgr` is retrieved from iObjSfdPanel. Then method `GetAvailableStandardPositior` called to get the standard positioning strategy names.

v. **Retrieves standard positioning strategy parameters**

```
...
    Dim PosStratParms As StrStandardPosStrategyParameters
    GetStdPosStrategyParams iObjSfdPanel, PosStratName, PosStratParms
...
```

In this step Uc retrieves the parameters of a particular standard positioning strategy. To do this Uc calls a method `GetStdPosStrategyParams`. This method and name of the standard positioing strategy as input parameter and it returns the collection of standard positioning strategy parameters as output parameter `PosStratParms` is a collection object of type `StrStandardPosStrategyParameters`, which is collection parameters required for defining positioing strate

The method `GetStdPosStrategyParams` is detailed as in the below sub steps.

i.
```
Sub GetStdPosStrategyParams(iObjSfdPanel As SfdPanel, iStdPosStrategyName As String, _
                            oStdPosStrategyParms As StrStandardPosStrategyParameters)
    'Get StrOpeningsMgr object
    Dim ObjStrOpeningsMgr As StrOpeningsMgr
    Set ObjStrOpeningsMgr = iObjSfdPanel.StrOpeningsMgr

    'Get standard positioining strategy parameters
    Set oStdPosStrategyParms = ObjStrOpeningsMgr.GetStandardPositioningStrategyParms(iStdPosStrategyName)
    If oStdPosStrategyParms.Count = 0 Then
        Err.Raise 1, Err.Source, "Positioning strategy parameters cannot found"
        Exit Sub
    End If
End Sub
```

In this method object `ObjStrOpeningsMgr` of type `StrOpeningsMgr` is retrieved from iObjSfdPanel. Then method `GetStandardPositioningStrate` get the positioning strategy parameters. Method `GetStandardPositioningStrategyParms` takes positioning strategy name as input parameter. Posit parameters related to this name will be returned in `oStdPosStrategyParms`.

vi. **Sets standard positioing strategy parameter's data**

```
...
    SetStdPosStrategyParamsDataForOffsetOffset PosStratParms
...
```

In this step Uc sets the data of "offset/offset" positioning strategy parameters by calling method `SetStdPosStrategyParamsDataForOffsetOffset`. Metho `SetStdPosStrategyParamsDataForOffsetOffset` takes positioning strategy parameters collection object(`PosStratParms`) as input parameters.

vii. **Create an opening and set its type, category, standard opening parameters**

```
...
    CreateOpeningAndSetData iObjSfdPanel, catStrOpeningModeStandard, "SldOpening", ContourName, StdContourParms, PosStratName
                            oObjStrOpening
...
```

In this step Uc creates an opening with no data set. Then it sets some of its data and returns created opening in `oObjStrOpening` as output parameter. First p which opening will be created, second parameter is opening type (catStrOpeningModeStandard is for standard opening), third parameter is category, fourth name, fifth parameter is contour parameters, sixth parameter is positioning strategy name and seventh parameter is positioning strategy parameters.

viii. **Updates created opening object**

```
...
    ObjPart.UpdateObject oObjStrOpening
End Sub
```

Method `UpdateObject` updates the created opening.

**Detailed steps of methods called in Uc**

- **SetRectContourParamsData method**

This method sets width, height and corner radius for Rectangle type of contour. This method takes collection of contour parameters, width, height and corner radiu

1. **Retrieves total number of contour parameters**

```
Sub SetRectContourParamsData(iContourParms As StrStandardContourParameters, iStrW As String, iStrH As String, iStrCR As Strin
    Dim NbOfContourParams As Long
    NbOfContourParams = iContourParms.Count
...
```

Here size of the of the contour parameters is retrieved.

2. **Retrieves a parameter from collection and set width/height/CornerRadius**

```
...
    For i = 1 To NbOfContourParams
      'Get a StrParameter from the collection
      Dim ObjStrParameter As StrParameter
      Set ObjStrParameter = iContourParms.Item.(i)
      'Get role of the parameter
      StrRole = ObjStrParameter.Role
      'Get parameter from StrParameter
      Dim ContourParam As Parameter
      Set ContourParam = ObjStrParameter.Parameter

      'Sets width
      If StrRole = "Str_Width" Then
        ContourParam.ValuateFromString (iStrW)
      End If

      'Sets height
      If StrRole = "Str_Height" Then
        ContourParam.ValuateFromString (iStrH)
      End If

      'Sets corner radius
      If StrRole = "Str_CornerRadius" Then
        ContourParam.ValuateFromString (iStrCR)
      End If
    Next
End Sub
```

Here collection object `iContourParms` which is of type `StrStandardContourParameters` is collection of `StrParameter`. So here in for loop Uc retrieves a role and parameter is retrieved and by checking its role(width/height/CornerRadius) its value is set.

- **SetStdPosStrategyParamsDataForOffsetOffset method**

This method sets U, V references and angle parameter value for "offset/offset" positioning strategy. This method takes collection of positioning strategy parameter (`StrStandardPosStrategyParameters`) as input parameters.

1. **Retrieves the number of Standard Positioning Strategy Parameters**

```
Sub SetStdPosStrategyParamsDataForOffsetOffset(iStdPosStrategyParms As StrStandardPosStrategyParameters)
    Dim SizeOfStdPosStratParms As Long
    SizeOfStdPosStratParms = iStdPosStrategyParms.Count
...
```

Here size of the collection of positioning strategy parameters is retrieved.

2. **Retrieves a positioning strategy parameter from the list**

```
...
    For i = 1 To SizeOfStdPosStratParms
        Set ObjStrStdPosStParam = iStdPosStrategyParms.Item(i)
...
```

Here a For loop is running through the collection of positioning strategy parameters. A positioning strategy parameter is retrieved in `ObjStrStdPosStParam`

3. **Sets U and V references**

```
...
        If (TypeName(ObjStrStdPosStParam) = "StrRefOffset") Then
            'Retrieves role of the parameter
            StrRole = ObjStrStdPosStParam.Role
            'Sets reference for U
            If StrRole = "PosSpecUCurve" Then
              Set ObjRefUShift = Manager.GetReferencePlane(ObjPart, 3, "LONG.-7")
              Set RefUShift = ObjPart.CreateReferenceFromObject(ObjRefUShift)
              ObjStrStdPosStParam.SetSpecification Nothing, RefUShift
              'set relevant side to Port
              ObjStrStdPosStParam.SetRelevantSide (4)
            End If
            'Sets reference for V
            If StrRole = "PosSpecVCurve" Then
              Set ObjRefVShift = Manager.GetReferencePlane(ObjPart, 2, "CROSS.94")
              Set RefVShift = ObjPart.CreateReferenceFromObject(ObjRefVShift)
              ObjStrStdPosStParam.SetSpecification Nothing, RefVShift
              'set relevant side to Fore
              ObjStrStdPosStParam.SetRelevantSide (2)
              'Set offset
              Set OffsetParm = ObjStrStdPosStParam.GetOffsetParm
              OffsetParm.ValuateFromString ("1000mm")
            End If
        End If
...
```

Here if parameter is of type `StrRefOffset` then U and V references are set. First role of the parameter is retrieved.

If the role is of type **"PosSpecUCurve"** then it is U reference. To set the U reference, first the reference to the plane LONG.-7 is created and then it is set by `SetSpecification`. After setting the U reference its side is set to Port side by calling method `SetRelevantSide(4)`. Here 4 means the port side.

If the role is of type **"PosSpecVCurve"** then it is V reference. To set the V reference, first the reference to the plane CROSS.94 is created and then it is set `SetSpecification`. After setting the V reference its side is set to Fore side by calling method `SetRelevantSide(2)`. Here 2 means the Fore side. To set th reference, first offset parameter is retrieved by calling method `GetOffsetParm` then its value set to 1000mm.

4. **Sets Angle parameter's value**

```
...
        If (TypeName(ObjStrStdPosStParam) = "StrPosAxisAdjustment") Then
```

```
            'get angle parameter and set its value
            Dim AngleParm As Parameter
            Set AngleParm = ObjStrStdPosStParam.GetAngleParameter
            AngleParm.ValuateFromString ("45deg")
        End If
    Next
End Sub
```

To set the angle first angle parameter is retrieved by using method `GetAngleParameter`. Then its value is set to 45deg.

- **CreateOpeningAndSetData method**

This method first creates opening with no data set. Then it sets its type, category, contour name, contour parameters, positioning strategy name and positioning str

1. **Retrieves StrOpenings and creates a opening with no properties set**

```
Sub CreateOpeningAndSetData(iObjSfdPanel As SfdPanel, iOpeningType As Long, iCategory As String, _
                            iContourName As String, iStdContourParms As StrStandardContourParameters, _
                            iPosStratName As String, iPosStratParms As StrStandardPosStrategyParameters, _
                            oObjStrOpening As StrOpening)
'   1- Retrieves StrOpenings and creates a opening with no properties set
    Set ObjStrOpenings = iObjSfdPanel.GetOpenings(0)
    Set oObjStrOpening = ObjStrOpenings.Add
...
```

To create opening first Uc retrieves the object `ObjStrOpenings` of type `StrOpenings`. Then `Add` method is called. This opening has no data set on it. Uc ne is opening type, category contour parameters etc.

2. **Retrieves StrCategoryMngt and sets category**

```
...
    Set ObjStrCategoryMngt = oObjStrOpening.StrCategoryMngt
    ObjStrCategoryMngt.SetCategory iCategory
...
```

In this step object `ObjStrCategoryMngt` is retrieved. Then method `SetCategory` is called to set the category. This method takes category as input paramete

3. **Sets opening type**

```
...
    oObjStrOpening.OpeningType = iOpeningType
...
```

In this step object opening type value is assigned to property `OpeningType`.

4. **Retrieves StrOpeningExtrusionMngt and sets forming extrusion mode**

```
...
    Set ObjStrOpeningExtrusionMngt = oObjStrOpening.StrOpeningExtrusionMngt
    ObjStrOpeningExtrusionMngt.ExtrusionMode = 1
...
```

In this step object forming mode is set. To set the forming mode, forming mode value is assigned to property `ExtrusionMode`. Here 1 defines Before Formi

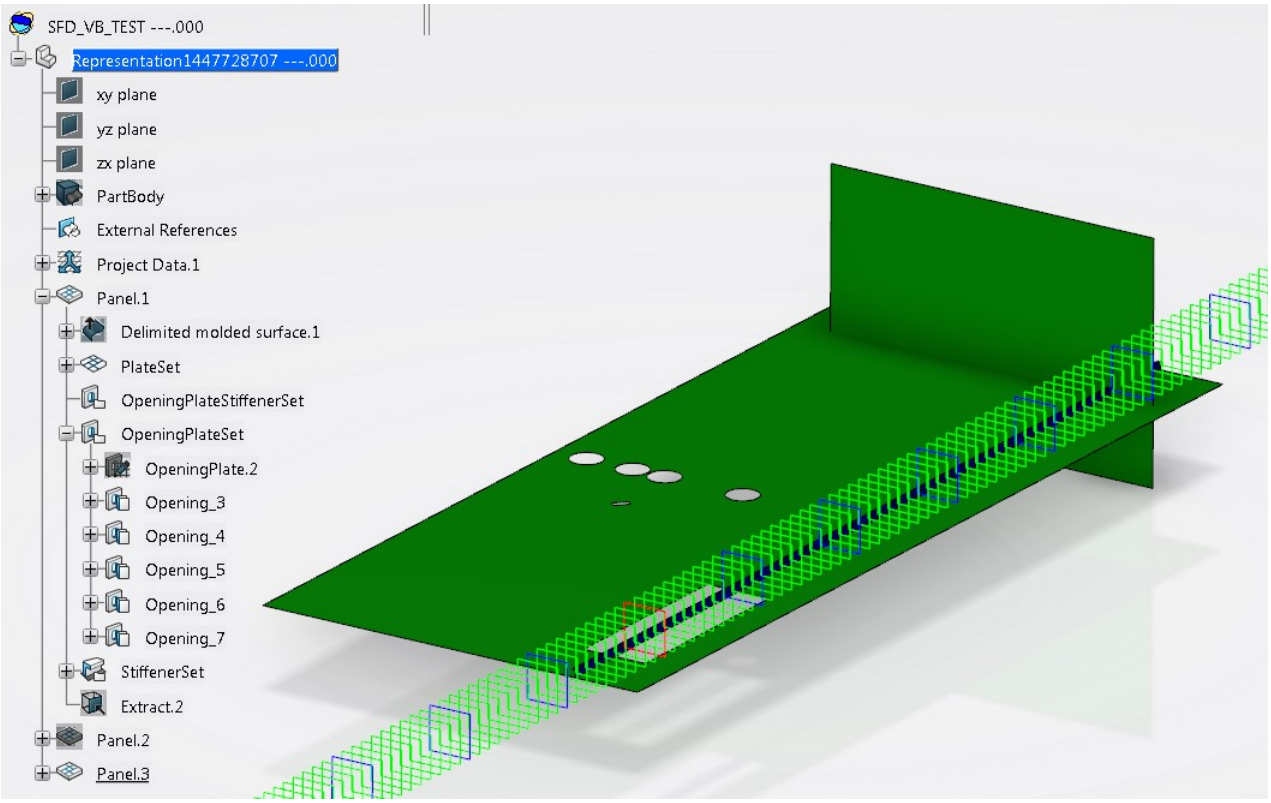
5. **Retrieves StrOpeningStandard and sets Direction, LimitMode and contour and positiong strategy parameters**

```
...
    'Get StrOpeningStandard object
    Dim ObjStrOpeningStandard As StrOpeningStandard
    Set ObjStrOpeningStandard = oObjStrOpening.StrOpeningStandard
    'set direction
    Set ObjDirectionRef = Manager.GetReferencePlane(ObjPart, 1, "DECK.1")
    Set DirectionRef = ObjPart.CreateReferenceFromObject(ObjDirectionRef)
    ObjStrOpeningStandard.Direction = DirectionRef
    'set limit mode
    ObjStrOpeningStandard.LimitMode = 0
    'set contour and positiong strategy parameters
    ObjStrOpeningStandard.SetContourAndPosStrategy iContourName, iStdContourParms, iPosStratName, iPosStratParms
End Sub
```

In this step object `ObjStrOpeningStandard` of type `StrOpeningStandard` is retrieved. Then reference to the DECK.1 plane is retrieved and it set as `Direc` property is set to 0. Here 0 means After forming mode. Then to set the contour name contour parameters, positioning strategy name and positioning strategy `SetContourAndPosStrategy` is called on object `ObjStrOpeningStandard`.

**Note:** The source also includes standard opening creation using positioning strategies Half Height/Offset, Mid Dist/Offset, Mid Dist/Mid Dist, Half Height/Mid Dist. T source go to CAAScdSfdUcCreatePanelSource.htm.

Fig.1: Standard Opening




# Navigating Structure Objects

This use case primarily focuses on the methodology to navigate structure objects.

Related Topics
Structure Functional Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcCGR.3dxml, CAAScdSfdUcSR.3dxml, CAAScdSfdUcSteel_A90.3dxml, CAAScdSfdUcNavigate.3dxml`, files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: CAAScdSfdUcNavigateSource.htm

This use case can be divided in nine steps:

1. Searches and opens model which is named as "SfdProduct"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves Transversal Bulkhead which is already created
5. Retrieving children of the panel
6. Retrieving Parents of the panel

1. **Searches and opens model which is named as "SfdProduct"**

   As a first step, the UC retrieves a model "SFD_VB_TEST" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_VB_TEST"
 OpenProduct prdName , SFDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor is returned in **`SFDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **`SFDPrdEditor`** is used.

```
...
    Set SFDProdSel = SFDPrdEditor.Selection
...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves Transversal Bulkhead which is already created**

In this step UC retrieves Transversal bulkhead which is already created in the model.

```
...
  Set RefSfdPanel = ObjPart.FindObjectByName("Transverse Bulkhead_1")
  SFDProdSel.Add RefSfdPanel
  Dim ObjStrNavigate As StrNavigate
  Set ObjStrNavigate = SFDProdSel.FindObject("CATIAStrNavigate")
...
```

5. **Retrieving children of the panel**

   In this step UC retrieves all children of panel which is retrieved in above step.

```
...
  Dim ListOfPanelChildren As References
  Set ListOfPanelChildren = ObjStrNavigate.GetChildren
...
```

6. **Retrieving Parents of the panel**

   In this step UC retrieves all parents of panel which is retrieved in step 4.

```
...
   Dim ListOfPanelParents As References
   Set ListOfPanelParents = ObjStrNavigate.GetParents
...
```

# Creating an SFD Parametric Panel

This use case primarily focuses on the methodology to initialize the SFD system and create SFD parametric panel.

Related Topics
Structure Functional Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcCGR.3dxml`, `CAAScdSfdUcSR.3dxml`, `CAAScdSfdUcSteel_A42.3dxml`, `CAAScdSfdUcWT18x179_5.3dxml`, `CAAScdSfdUcCreateSketchBasedPanel_RefSkt.3dxml`, `Bracket_KN6.3dxml`, and `CAAScdSfdUcCreateSketchBasedPanel.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: CAAScdSfdUcCreateSketchBasedPanelSource.htm

This use case can be divided in 12 steps:

1. Searches and opens model which is named as "SFD_Product"
2. Navigates to the part object and retrieves the part object
3. Retrieves Service manager (RfgService)
4. Retrieves SfdFactory
5. Retrieves a SFD panels and create stiffeners
6. Adds a geometrical set
7. Creates a Parametric Panel using LONG.9 as support plane
8. Updates the part
9. Detailed steps of subroutines called in Uc

1. **Searches and opens model which is named as "SFD_Product"**

   As a first step, the UC retrieves a model "SFD_Product" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_Product"
 OpenProduct prdName , SFDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor i returned in **`SFDPrdEditor`**.

2. **Navigates to the part object and retrieves the part object**

   In this step, starting from the root product, the UC navigates to the part object of the SFD system and retrieves a `Reference` on it.

```
...
  Set SFDProdSel = SFDPrdEditor.Selection
  Dim product1Service   As PLMProductService
  Set product1Service = SFDPrdEditor.GetService("PLMProductService")
  Dim ObjVPMRootOccurrence   As VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference As VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances As VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjVPMRepReference.GetItem("Part")
  If ObjPart Is Nothing Then
        Err.Raise 1, Err.Source, "Cannot retrieve part"
    Exit Sub
  End If
        ...
```

3. **Retrieves Service manager (RfgService)**

   In this step UC retrieves `RfgServices`.

```
...
```

```
  Set Manager = CATIA.ActiveEditor.GetService("RfgServices")
...
```

`GetService` method returns `RfgServices`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

4. **Retrieves SfdFactory**

   In this step Uc initializes resources for SFD.

```
...
  Dim ObjSfdFactory As SfdFactory
  Set ObjSfdFactory = ObjPart.GetCustomerFactory("SfdFactory")
...
```

   In this step Uc retrieves object `ObjSfdFactory` of type `SfdFactory`. It is retrieved by using `GetCustomerFactory` method.

5. **Retrieves a SFD panels and create stiffeners**

   In this step Uc retrieves SFd panels and creates stiffener on it.

```
...
  ' 5- Retrieves a SFD panels and create stiffeners
  Set RefSfdPanel = ObjPart.FindObjectByName("Transverse Bulkhead_3")
  Dim ObjSfdTransvPanel As SfdPanel
  SFDProdSel.Add RefSfdPanel
  Set ObjSfdTransvPanel = SFDProdSel.FindObject("CATIASfdPanel")
  CreateStiffener ObjSfdTransvPanel, ObjSfdTransvStiffener
  Set RefSfdPanel = ObjPart.FindObjectByName("Deck_1")
  Dim ObjSfdDeckPanel As SfdPanel
  SFDProdSel.Add RefSfdPanel
  Set ObjSfdDeckPanel = SFDProdSel.FindObject("CATIASfdPanel")
  CreateStiffener ObjSfdDeckPanel, ObjSfdLongStiffener
...
```

   In this step Uc retrieves object `ObjSfdTransvPanel` & `ObjSfdDeckPanel` of type `SfdPanel` and it creates stiffeners on these panels using `CreateStiffener` method.

6. **Adds a geometrical set**

   Call `AddGeometricalSet` method to create a new geometrical set under the SFD system. `AddGeometricalSet` method takes a part as input parameter and gives a reference to the newly created geometrical set as the output.

```
...
  Dim RefGeometricalSet As Reference
  AddGeometricalSet ObjPart, RefGeometricalSet
...
```

   The method `AddGeometricalSet` is detailed as in the below sub steps.

   i. **Obtains HybridBodies from Part**

```
  Sub AddGeometricalSet(iObjPart As Part, oRefGeometricalSet As Reference)
    ' Add a Geometrical Set in Part for creating Sketch Based Panel
    Dim oHybridBodies As HybridBodies
    Set oHybridBodies = iObjPart.HybridBodies
...
```

      Here `HybridBodies` method returns the HybridBodies object in the part.

   ii. **Adds a new HybridBody**

```
...
    Dim oGeometricalSet As HybridBody
    Set oGeometricalSet = oHybridBodies.Add()
...
```

   iii. **Obtains a reference on the new geometric set and returns**

```
...
    Set oRefGeometricalSet = iObjPart.CreateReferenceFromObject(oGeometricalSet)
    If oRefGeometricalSet Is Nothing Then
      Err.Raise 1, Err.Source, "Cannot Create GeometricalSet"
      Exit Sub
    End If
  End Sub
```

      This step sets the newly created geometric set as the Active Object. The `SfdSketchBasedPanel` will be created under this geometric set.

7. **Creates a Parametric Panel using LONG.9 as support plane**

   Call `CreateSketchBasedPanel` method to create a Parametric Panel. A geomtric set is an input to the method `CreateSketchBasedPanel`. The Parametric Panel i created under this geometric set. The second input to this method is the SfdFactory using which the Parametric Panel is create. The method returns created parame panel as output parameter in `ObjSfdSketchBasedPanel`.

```
...
  Dim ObjSfdSketchBasedPanel As SfdSketchBasedPanel
  CreateSketchBasedPanel RefGeometricalSet, ObjSfdFactory, ObjSfdSketchBasedPanel
...
```

   The method `CreateSketchBasedPanel` is detailed as in the below sub steps.

   i. **Creates sketch based panel with no properties**

```
Sub CreateSketchBasedPanel(iDestPart As Reference, iObjSfdFactory As SfdFactory, oObjSfdSketchBasedPanel As SfdSketchBasedPan

    ' Call subroutine AddSketchBasedPanel to add a new Sketch Based Panel using the SFD Factory
```

```
    AddSketchBasedPanel iDestPart, iObjSfdFactory, oObjSfdSketchBasedPanel
...
```

Follow the link for details of what is done in the `AddSketchBasedPanel` method.

ii. **Sets the category**

```
...
    ' Set the Category
    SetCategory oObjSfdSketchBasedPanel, "ParametricPanel"
...
```

Follow the link for details of what is done in the `SetCategory` method.

iii. **Set LONG.9 as the support plane**

```
...
    ' Browse to "LONG.9" reference plane and set it as the support plane
    Dim PanelSupport As Reference
    Set PanelSupport = Manager.GetReferencePlane(ObjPart, 3, "LONG.9")
    SetSketchBasedPanelSupport oObjSfdSketchBasedPanel, PanelSupport
...
```

Follow link to know how the subroutine `SetSketchBasedPanelSupport` obtains the `StrPanelSurf` and sets `LONG.9` as the support plane.

iv. **Set the reference sketch for the parametric panel**

```
...
    ' Set the DMS
    Dim PanelDMS As StrSketchBasedDMSMngt
    Set PanelDMS = oObjSfdSketchBasedPanel.StrSketchBasedDMSMngt
    PanelDMS.SetStrSketch ("SAMPLE_RCO_2LIMITS_KB")
...
```

The `StrSketchBasedDMSMngt` is obtained from `SfdSketchBasedPanel`. Further, the method `SetStrSketch` is called to set the reference sketch on `SfdSketchBasedPanel`. The **"SAMPLE_RCO_2LIMITS_KB"** is a user defined sketch retrieved from the DB. The parametric panel is created based on t sketch.

v. **Set the public parameters of the reference sketch**

```
...
    Dim ObjStrReferenceSketchPublicParms As StrReferenceSketchPublicParameters
    Set ObjStrReferenceSketchPublicParms = oObjSfdSketchBasedPanel.StrReferenceSketchPublicParameters
    Dim NSize As Long
    NSize = ObjStrReferenceSketchPublicParms.Count
    Dim ObjPublicParm As Parameter
    Set ObjPublicParm = ObjStrReferenceSketchPublicParms.Item(2)
    ObjPublicParm.ValuateFromString ("2000mm")
...
```

The reference sketch used in this UC has two public parameters. The UC modifies only the second public parameter and sets its value to 2000mm. The `StrReferenceSketchPublicParameters` object is obtained from `oObjSfdSketchBasedPanel` and the `Item` method is called to obtain the public paramete index 2.

vi. **Set the material, the material throw orientation and the thickness of the parametric panel**

```
...
    ' Set material
    SetMaterial oObjSfdSketchBasedPanel, "Steel A42"
    ' Set material throw orientation
    Dim ExtrusionManager As StrPlateExtrusionMngt
    Set ExtrusionManager = oObjSfdSketchBasedPanel.StrPlateExtrusionMngt
    ExtrusionManager.ThrowOrientation = 1
    ' Set thickness of the parametric panel
    Dim Thickness As Parameter
    Set Thickness = ExtrusionManager.GetThickness
    Thickness.ValuateFromString "12mm"
...
```

The UC calls subroutine `SetMaterial` to set the material. To know what is done withing, follow the link.
The `StrPlateExtrusionMngt` object is obtained from `oObjSfdSketchBasedPanel` object. The method `ThrowOrientation` on `StrPlateExtrusionMngt` object in turn sets the material throw orientation.
Further, the thickness `Parameter` is obtained for the thickness and set to **"12mm"**

vii. **Set the other necessary limits to position the parametric panel**

```
...
    ' Set panel limits
    SetSketchBasedPanelLimits oObjSfdSketchBasedPanel
...
```

Here, the UC positions the parametric panel using the two stiffeners, **"Transverse Bulkhead Vertical Stiffener_2"** and **"Deck Longitudinal Stiffener_1"** present in the data as the limits.

- The `SetSketchBasedPanelLimits` subroutine is detailed below:

```
...
    Sub SetSketchBasedPanelLimits(iObjSfdSketchBasedPanel As SfdSketchBasedPanel)
      ' Obtain the StrPanelLimitMngt
      Dim ObjSfdPanelLimitMngt As StrPanelLimitMngt
      Set ObjSfdPanelLimitMngt = iObjSfdSketchBasedPanel.StrPanelLimitMngt
      If ObjSfdPanelLimitMngt Is Nothing Then
        Err.Raise 1, Err.Source, "Cannot retrieve StrPanelLimitMngt object"
        Exit Sub
      End If

      ' Get Reference to an existing SFD Stiffener
      Set Limit1 = ObjPart.CreateReferenceFromObject(ObjSfdTransvStiffener)
```

```
            ' Get Reference to another existing SFD Stiffener
            Set Limit2 = ObjPart.CreateReferenceFromObject(ObjSfdLongStiffener)

            ' Apply the References to the 2 SFD stiffeners as limits to the Skecth Based Panel
            ObjSfdPanelLimitMngt.SetLimitingObject Limit1, 2, 0, -1
            ObjSfdPanelLimitMngt.SetLimitingObject Limit2, 3, 0, -1

        End Sub
    ...
```

The `SetSketchBasedPanelLimits` subroutine takes `SfdSketchBasedPanel` object as input.
Internally, it obtains the `StrPanelLimitMngt` from `SfdSketchBasedPanel` and calls the `SetLimitingObject` method on it.
The two stiffeners which are set as limits are obtained by calling the `FindObjectByName` on the part.
The parameters to `SetLimitingObject` are "limiting object", "limit index", "orientation" and "limit type" in that order.

viii. **Calls method Run to complete the creation os parametric panel**

```
    ...
        ' Call run to create the parametric panel
        Run oObjSfdSketchBasedPanel
    End Sub
```

8. **Update the part**

```
...
  ' Update the part
  ObjPart.Update
End Sub
```

**Detailed steps of subroutines called in Uc**

- **AddSketchBasedPanel**

This method adds a new empty sketch based panel. It takes the `SfdFactory` and a reference to the destination part where the sketch based panel will be created. In this UC we create the sketch based panel under the new geometric set.
This subroutine returns `SfdSketchBasedPanel`, the empty sketch based panel.

```
Sub AddSketchBasedPanel(iDestPart As Reference, iObjSfdFactory As SfdFactory, oObjSfdSketchBasedPanel As SfdSketchBasedPanel)
  Set oObjSfdSketchBasedPanel = iObjSfdFactory.AddSketchBasedPanel(iDestPart, 5)
  If oObjSfdSketchBasedPanel Is Nothing Then
    Err.Raise 1, Err.Source, "Cannot Create SfdSketchBasedPanel"
    Exit Sub
  End If
End Sub
```

The second parameter to the `AddSketchBasedPanel` method of `SfdFactory` is the mode of creation of the parametric panel. In this UC we use mode **5** which is creation with two stiffeners as the limits.

- **SetCategory**

The `SetCategory` subroutine accepts a `SfdSketchBasedPanel` and a string referring to category name as input.

```
Sub SetCategory(iObjSfdSketchBasedPanel As SfdSketchBasedPanel, iStrCategory As String)
  Dim ObjSfdCategoryMngt As StrCategoryMngt
  Set ObjSfdCategoryMngt = iObjSfdSketchBasedPanel.StrCategoryMngt
  If ObjSfdCategoryMngt Is Nothing Then
      Err.Raise 1, Err.Source, "Cannot retrieve StrCategoryMngt object"
      Exit Sub
  End If

  ObjSfdCategoryMngt.AutomaticName = True
  ObjSfdCategoryMngt.SetCategory iStrCategory

End Sub
```

Internally, it obtains the `StrCategoryMngt` object from `SfdSketchBasedPanel` object and calls method `SetCategory` to set the category.

- **SetSketchBasedPanelSupport**

The `SetSketchBasedPanelSupport` subroutine accepts a `SfdSketchBasedPanel` and a `Reference` to support as input.

```
Sub SetSketchBasedPanelSupport(iObjSfdSketchBasedPanel As SfdSketchBasedPanel, iSupport As Reference)
  Dim ObjSfdPanelSupport As StrPanelSurf
  Set ObjSfdPanelSupport = iObjSfdSketchBasedPanel.StrPanelSurf
  If ObjSfdPanelSupport Is Nothing Then
      Err.Raise 1, Err.Source, "Cannot retrieve StrPanelSupport object"
      Exit Sub
  End If

  ObjSfdPanelSupport.Support = iSupport
End Sub
```

Internally, it obtains the `StrPanelSurf` object from `SfdSketchBasedPanel` object and calls method `Support` to set the panel support.

- **SetMaterial**

The `SetMaterial` subroutine accepts a `SfdSketchBasedPanel` and a string referring to material name as input.

```
Sub SetMaterial(iObjSfdSketchBasedPanel As SfdSketchBasedPanel, iStrMaterial As String)
  Dim ObjMaterialMngt As StrMaterialMngt
  Set ObjMaterialMngt = iObjSfdSketchBasedPanel.StrMaterialMngt
  If ObjMaterialMngt Is Nothing Then
      Err.Raise 1, Err.Source, "Cannot retrieve StrMaterial object"
      Exit Sub
  End If
  ObjMaterialMngt.SetMaterial iStrMaterial
End Sub
```

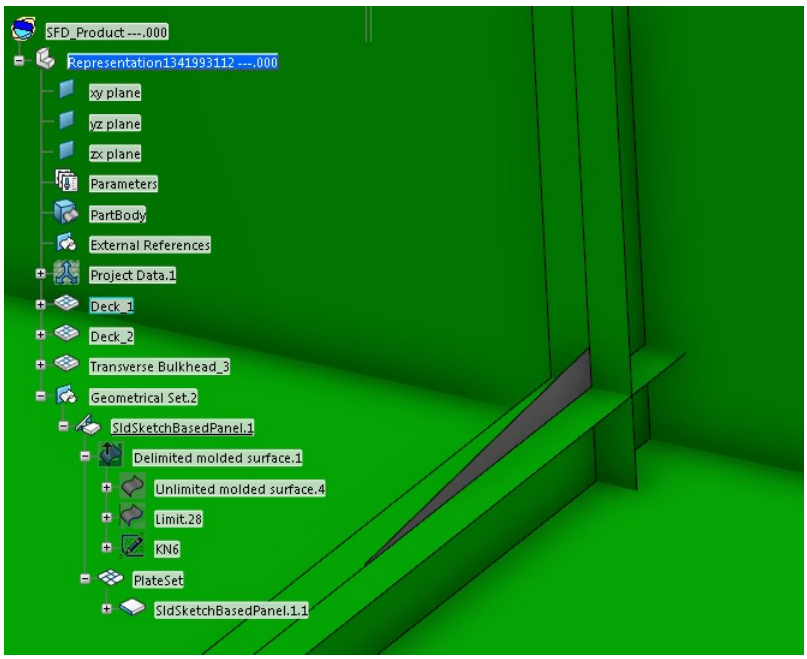
Internally, it obtains the `StrMaterialMngt` object from `SfdSketchBasedPanel` object and calls method `SetMaterial` to set the material.

- **Run**

The `SetMaterial` subroutine accepts a `SfdSketchBasedPanel` as input and after obtaining an object of `StrSfdPlatesMngt` from it, it calls the `Run` method on `StrSfdPlatesMngt`.

```
Sub Run(iObjSfdSketchBasedPanel As SfdSketchBasedPanel)
  ' Obtain the StrSfdPlatesMngt
  Dim ObjSfdPlatesMngt As StrSfdPlatesMngt
  Set ObjSfdPlatesMngt = iObjSfdSketchBasedPanel.StrSfdPlatesMngt
  If ObjSfdPlatesMngt Is Nothing Then
      Err.Raise 1, Err.Source, "Cannot retrieve StrSfdPlatesMngt object"
      Exit Sub
  End If

  ObjSfdPlatesMngt.Run
  'Panel is Created
End Sub
```

Fig.1: Parametric Panel




# Converting an SFD Stiffener

This use case primarily focuses on the methodology to convert SFD Stiffener into Web/Flange or Web/FlatBar.

Related Topics
Structure Functional Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcCGR.3dxml, CAAScdSfdUcSR.3dxml, CAAScdSfdUcWT18x179_5.3dxml, CAAScdSfdUcSteel_A90.3dxml, CAAScdSfdUcPart.3dxml, CAAScdSfdUcConvertStiffener.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: CAAScdSfdUcConvertStiffenerSource.htm

This use case can be divided in eight steps:

1. Searches and opens model which is named as "SFD_ConvertStiffener"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves Service manager
5. Retrieves SfdFactory for getting SFD objects
6. Retrieves a SFD panel("panel.1") object
7. Creates a Stiffener
8. Calling ConvertStiffener
9. Update the Part object

1. **Searches and opens model which is named as "SFD_ConvertStiffener"**

As a first step, the UC retrieves a model "SFD_ConvertStiffener" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_ConvertStiffener"
 OpenProduct prdName , SFDPrdEditor
...
```

The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor is returned in **`SFDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **`SFDPrdEditor`** is used.

```
...
   Set SFDProdSel = SFDPrdEditor.Selection
...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves Service Manager**

   In this step UC retrieves object of RfgService.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

   *GetService* method returns the service `RfgService` in `Manager` variable. This service provides various services.

5. **Retrieves SfdFactory for getting SFD objects**

   In this step UC retrieves SFDFactory for getting SFD objects.

```
...
  Set ObjSfdFactory = ObjPart.GetCustomerFactory("SfdFactory")
...
```

6. **Retrieves a SFD panel ("panel.1") object**

   In this step UC finds a SFD panel in the part and retrieve SfdPanel object using Selection object.

```
...
  Set RefSfdPanel = ObjPart.FindObjectByName("Panel.1")
  Dim ObjSfdPanel As SfdPanel
  SFDProdSel.Add RefSfdPanel
  Set ObjSfdPanel = SFDProdSel.FindObject("CATIASfdPanel")
...
```

   In above lines, *FindObjectByName* method finds object whose name is "Panel.1" and returns reference to it. Here `RefSfdPanel` is of type `Reference`. To retrieve `SfdPanel` object from the reference, add retrieved reference to the selection and call *FindObject* method as shown above. This will give the `SfdPanel` object.

7. **Creates a Stiffener**

   Now panel is available to create stiffener on it. Call `CreateStiffener` method to create stiffener. `CreateStiffener` method takes a panel as input and created stiffener is returned in output parameter.

```
...
  Dim ObjSfdStiffener As StrSfdStiffener
  CreateStiffener ObjSfdPanel, ObjSfdStiffener
...
```

   The method `CreateStiffener` is detailed as in the below sub steps.

   i. Retrieves the `StrSfdStiffeners` object from panel. Then `AddStiffener` method from object `StrSfdStiffeners` will create a new empty stiffener.

```
Sub CreateStiffener(iObjSfdPanel As SfdPanel, oObjSfdStiffener As StrSfdStiffener)
    'Get StrSfdStiffeners object
    Dim ObjSfdStiffeners As StrSfdStiffeners
    Set ObjSfdStiffeners = iObjSfdPanel.StrSfdStiffeners
    'Add Stiffener
    Set oObjSfdStiffener = ObjSfdStiffeners.AddStiffener
...
```

   `StrSfdStiffeners` object is retrieved in `ObjSfdStiffeners` variable from `iObjSfdPanel` object. On `ObjSfdStiffeners` object `AddStiffener` method is called to create the empty stiffener. Now Uc needs to set the different properties of the stiffener like material, category etc.

   ii. Set different properties of the stiffener like material. Retrieve the `StrMaterialMngt` object from `StrSfdStiffener` object and set the material for the stiffener using `SetMaterial` method.

```
...
    'Get StrMaterialMngt object
    Dim ObjMaterialMngt As StrMaterialMngt
    Set ObjMaterialMngt = oObjSfdStiffener.StrMaterialMngt
    'Set material of the stiffener
    ObjMaterialMngt.SetMaterial("Steel A90")
...
```

   `ObjMaterialMngt` object is of type `StrMaterialMngt`. It is retrieved from the stiffener `oObjSfdStiffener`. `SetMaterial` method is called to set the material on object `ObjMaterialMngt`.

   iii. To set the category retrieve the `StrCategoryMngt` object from `StrSfdStiffener` object and set category using `SetCategory` method.

```
...
    'Get StrCategoryMngt object
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = oObjSfdStiffener.StrCategoryMngt
    'Set category of the stiffener
    ObjStrCategoryMngt.SetCategory("SldStiffener")
```

```
...
```

iv. Sets the `ProfileType` property of the stiffener to catStrProfileModeSurfSurf (here catStrProfileModeSurfSurf means profile is created with the intersection of two surfaces).

```
...
    'Set type of the stiffener
    oObjSfdStiffener.ProfileType = catStrProfileModeSurfSurf
...
```

v. Retrieves the `StrSectionMngt` object from the created stiffener object and sets the different section parameters like section name, anchor point, web orientation, flange orientation.

```
...
    'Get StrSectionMngt object
    Dim ObjStrSectionMngt As StrSectionMngt
    Set ObjStrSectionMngt = oObjSfdStiffener.StrSectionMngt
    'Set different section parameters
    ObjStrSectionMngt.SetSectionName ("WT18x179.5")
    ObjStrSectionMngt.AnchorPoint = "catStrTopCenter"
    ObjStrSectionMngt.WebOrientation = 1
    ObjStrSectionMngt.FlangeOrientation = 1
    ObjStrSectionMngt.AngleMode = 0
...
```

`ObjStrSectionMngt` object which is of type `StrSectionMngt` is used to set the different section properties of the stiffener. `SetSectionName` method sets the section name for the stiffener. `AnchorPoint` property is used to set anchor point to "catStrTopCenter". Similarly `WebOrientation` and `FlangeOrientation` properties are set to 1.

vi. Retrieves the `StrProfileLimitMngt` object from the created stiffener object and sets the Start limit and End limit of the stiffener.

```
...
    'Get StrProfileLimitMngt object
    Dim ObjStrProfileLimitMngt As StrProfileLimitMngt
    Set ObjStrProfileLimitMngt = oObjSfdStiffener.StrProfileLimitMngt
    'Set the profile limits
    Set ObjStartLimit = Manager.GetReferencePlane(ObjPart, 2, "CROSS.70")
    Set StartLimit = ObjPart.CreateReferenceFromObject(ObjStartLimit)
    ObjStrProfileLimitMngt.SetLimitingObject 1, StartLimit
    Set ObjEndLimit = Manager.GetReferencePlane(ObjPart, 2, "CROSS.110")
    Set EndLimit = ObjPart.CreateReferenceFromObject(ObjEndLimit)
    ObjStrProfileLimitMngt.SetLimitingObject 2, EndLimit
...
```

`ObjStrProfileLimitMngt` is retrieved from the `oObjSfdStiffener` object. Variable `StartLimit` and `EndLimit` are of type `Reference` which contains references to planes "CROSS.70" and "CROSS.110" respectively. Reference to plane is retrieved using `GetReferencePlane` method on `Manager` object. `Manager` object is of type `RfgService`. `ObjStrProfileLimitMngt` calls `SetLimitingObject` method to set the start and end limit of the of the stiffener. `SetLimitingObject` method hass two input parameters. First input defines whether it is start of the stiffener or end of the stiffener. Second input is the limiting object.

vii. Retrieves the `StrProfileSurfSurf` object from the created stiffener object and sets two surfaces which are intersecting with each other. At the intersection of these two surfaces stiffener will be created.

```
...
    'Get StrProfileSurfSurf object
    Dim ObjStrProfileSurfSurf As StrProfileSurfSurf
    Set ObjStrProfileSurfSurf = oObjSfdStiffener.StrProfileSurfSurf
    'Set two surfaces
    Dim DMS As Reference
    Set DMS = iObjSfdPanel.GetDelimitedSupport
    ObjStrProfileSurfSurf.FirstSurface = DMS
    Set ObjWebSupport = Manager.GetReferencePlane(ObjPart, 3, "LONG.4")
    Set WebSupport = ObjPart.CreateReferenceFromObject(ObjWebSupport)
    ObjStrProfileSurfSurf.SecondSurface = WebSupport
...
```

`ObjStrProfileSurfSurf` object is retrieved from `oObjSfdStiffener` object. Here Uc is setting two intersecting surfaces for stiffener creation. To set first surface, it retrieves delimited surface of the `iObjSfdPanel` using method `GetDelimitedSupport` in `DMS` object. Then `DMS` is assigned to the `FirstSurface` property of the `ObjStrProfileSurfSurf` object. Similarly `SecondSurface` is set.

viii. Updates the created stiffener.

```
...
    ObjPart.UpdateObject oObjSfdStiffener
End Sub
...
```

8. **Calling ConvertStiffener**

```
...
  Dim ObjSfdConvertStiffener As SfdConvertStiffener
  ConvertStiffener ObjSfdConvertStiffener,ObjSfdStiffener
...
```

`oObjSfdConvertStiffener` object is retreived from `iObjSfdStiffener` object.

```
...
  Set oObjSfdConvertStiffener = iObjSfdStiffener.SfdConvertStiffener

    'Set Mode
    oObjSfdConvertStiffener.Mode = 1
    'Set Category
    'Example:: For FlatBar: Category - "StiffenerOnFreeEdge"
    oObjSfdConvertStiffener.Category "Panel", "Panel"
    'Set Material
    oObjSfdConvertStiffener.Material "Steel A90", "Steel A90"
    'Set Panel Names
    oObjSfdConvertStiffener.PanelNames "Panel", "Flange"
    'Set Stiffener State
```

```
    oObjSfdConvertStiffener.RemoveStiffener FALSE
    'Set Flange State
    oObjSfdConvertStiffener.FlangeState TRUE
    'Set Flange Section Name
    oObjSfdConvertStiffener.FlatBarSectionName "WT18x179.5"
    'ConvertStiffener
    oObjSfdConvertStiffener.ConvertStiffener
...
```

`Mode`: Setting up the Mode for type of conversion. 1- Web/Flange 2- Web/FlatBar

`Category`: Setting up the Category for Web and Flange or FlatBar.

`Material`: Setting up the Material for Web and Flange or FlatBar.

`PanelNames`: Setting up the Panel Names for Web and Flange or FlatBar.

`RemoveStiffener`: Setting up whether want to remove stiffener or not after it's conversion.

`FlangeState`: Setting up whether want to convert flange or not.

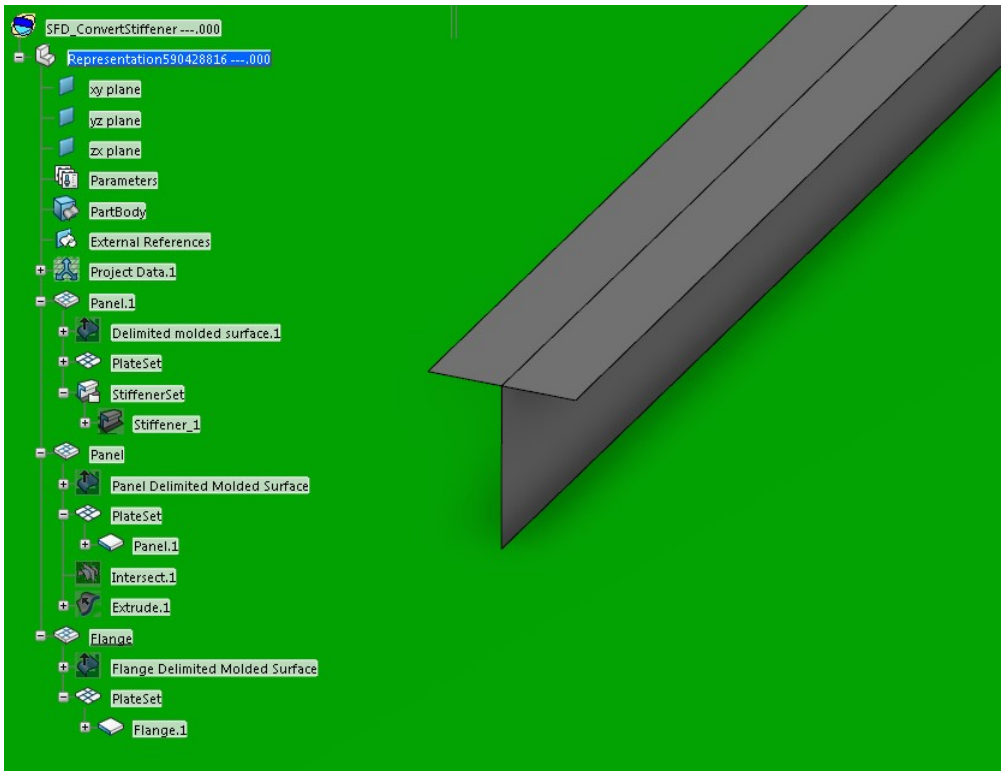
`FlatBarSectionName`: Setting up the section name for FlatBar.

`ConvertStiffener`: Calling ConvertStiffener for conversion after setting above parameters.

9. **Update the `Part` object**

```
...
    ObjPart.UpdateObject ObjSfdConvertStiffener
End Sub
```

Fig.1: Converting Stiffener to Web/Flange




# Creating an SFD Flange

This use case primarily focuses on the methodology to create SFD flange on Plate.

---

Related Topics
Structure Functional Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcCGR.3dxml, CAAScdSfdUcSR.3dxml, CAAScdSfdUcSteel_A42.3dxml, CAAScdSfdUcPart.3dxml, CAAScdSfdUcFlange.3dxml` and files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: CAAScdSfdUcCreateFlangeSource.htm

This use case can be divided in eight steps:

1. Searches and opens model which is named as "SFD_Flange"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves Service manager
5. Retrieves SfdFactory for getting SFD objects

6. Retrieves a SFD panel("Panel.1.1") object
7. Create Flange
8. Update the Part object

1. **Searches and opens model which is named as "SFD_Flange"**

   As a first step, the UC retrieves a model "SFD_Flange" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_Flange"
 OpenProduct prdName , SFDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor is returned in **`SFDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **`SFDPrdEditor`** is used.

```
 ...
    Set SFDProdSel = SFDPrdEditor.Selection
 ...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves Service Manager**

   In this step UC retrieves object of RfgService.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

   *GetService* method returns the service `RfgService` in `Manager` variable. This service provides various services.

5. **Retrieves SfdFactory for getting SFD objects**

   In this step UC retrieves SFDFactory for getting SFD objects.

```
...
  Set ObjSfdFactory = ObjPart.GetCustomerFactory("SfdFactory")
...
```

6. **Retrieves a SFD plate ("Panel.1.1") object**

   In this step UC finds a SFD plate in the part and retrieve StrSfdPlate object using Selection object.

```
...
  Set RefSfdPlate = ObjPart.FindObjectByName("Panel.1.1")
  Dim ObjSfdPlate As StrSfdPlate
  SFDProdSel.Add RefSfdPlate
  Set ObjSfdPlate = SFDProdSel.FindObject("CATIASfdPlate")
...
```

   In above lines, *FindObjectByName* method finds object whose name is "Panel.1.1" and returns reference to it. Here `RefSfdPanel` is of type `Reference`. To retrieve `StrSfdPanel` object from the reference, add retrieved reference to the selection and call *FindObject* method as shown above. This will give the `StrSfdPanel` object.

7. **Create Flange**

   Now plate is available to create flange on it. Call `CreateFlange` method to create flange. `CreateFlange` method takes a plate as input and created flange is returned in output parameter.

```
...
  Dim ObjStrFlange As StrFlange
  CreateFlange ObjSfdPlate, ObjStrFlange
...
```

   `oObjStrFlange` object is retreived from `AddFlange`.

```
Sub AddFlange(iObjSfdPlate As StrSfdPlate, oObjStrFlange As StrFlange)

  'Get StrFlanges object
  Dim ObjStrFlanges As StrFlanges
  Set ObjStrFlanges = iObjSfdPlate.Flanges

  'Create a flange
  Set oObjStrFlange = ObjStrFlanges.Add

End Sub
```

   `StrFlanges` object is retrieved in `ObjStrFlanges` variable from `iObjSfdPlate` object. On `ObjStrFlanges` object **`Add`** method is called to create the empty flange. Now Uc needs to set the different properties of the flange like Type, WidthMeasurementType, Start Limit, End Limit, etc.

```
...
  'Add Flange
   AddFlange iObjSfdPlate, oObjStrFlange
```

```
    'Set Type
    oObjStrFlange.Type = 1

    'Set FlangeStartLimit
    Set RefSfdPlane = ObjPart.FindObjectByName("CROSS.40")
    Dim ObjSfdReferencePlane_1 As Reference
    SFDProdSel.Add RefSfdPlane
    Set ObjSfdReferencePlane_1 = SFDProdSel.FindObject("CATIAReference")
    oObjStrFlange.FlangeStartLimit = ObjSfdReferencePlane_1

    'Set FlangeEndLimit
    Set RefSfdPlane = ObjPart.FindObjectByName("CROSS.24")
    Dim ObjSfdReferencePlane_2 As Reference
    SFDProdSel.Add RefSfdPlane
    Set ObjSfdReferencePlane_2 = SFDProdSel.FindObject("CATIAReference")
    oObjStrFlange.FlangeEndLimit = ObjSfdReferencePlane_2

    'Set WidthMeasurementType
    oObjStrFlange.WidthMeasurementType = 1

    'Get OperatedPlate
    Dim ObjSfdPlate_1 As Reference
    Set ObjSfdPlate_1 = oObjStrFlange.OperatedPlate

    'Get Type
    Dim oType As long
    oType = oObjStrFlange.Type

    'Get BendingAngle
    Dim oBendingAngle As Parameter
    Set oBendingAngle = oObjStrFlange.BendingAngle
    oBendingAngle.ValuateFromString("120deg")

    'Get BendingRadius
    Dim oBendingRadius As Parameter
    Set oBendingRadius = oObjStrFlange.BendingRadius
    oBendingRadius.ValuateFromString("8mm")

    'Get FlangeStartLimit
    Dim oFlangeStartLimit As Reference
    Set oFlangeStartLimit = oObjStrFlange.FlangeStartLimit

    'Get FlangeEndLimit
    Dim oFlangeEndLimit As Reference
    Set oFlangeEndLimit = oObjStrFlange.FlangeEndLimit

    'Get FlangeWidth
    Dim oFlangeWidth As Parameter
    Set oFlangeWidth = oObjStrFlange.FlangeWidth
    oFlangeWidth.ValuateFromString("500mm")

    'Get WidthMeasurementType
    Dim oWidthMeasurementType As long
    Set oWidthMeasurementType = oObjStrFlange.WidthMeasurementType

    'Get StartEndCutOffset
    Dim oStartEndCutOffset As Parameter
    Set oStartEndCutOffset = oObjStrFlange.StartEndCutOffset
    oStartEndCutOffset.ValuateFromString("10mm")

    'Get EndEndCutOffset
    Dim oEndEndCutOffset As Parameter
    Set oEndEndCutOffset = oObjStrFlange.EndEndCutOffset
    oEndEndCutOffset.ValuateFromString("10mm")

    'Get StartEndCutRadius
    Dim oStartEndCutRadius As Parameter
    Set oStartEndCutRadius = oObjStrFlange.StartEndCutRadius
    oStartEndCutRadius.ValuateFromString("50mm")

    'Get EndEndCutRadius
    Dim oEndEndCutRadiusAs Parameter
    Set oEndEndCutRadius= oObjStrFlange.EndEndCutRadius
    oEndEndCutRadius.ValuateFromString("50mm")

    'Get StartEndCutDistance
    Dim oStartEndCutDistance As Parameter
    Set oStartEndCutDistance = oObjStrFlange.StartEndCutDistance
    oStartEndCutDistance.ValuateFromString("100mm")

    'Get EndEndCutDistance
    Dim oEndEndCutDistance As Parameter
    Set oEndEndCutDistance = oObjStrFlange.EndEndCutDistance
    oEndEndCutDistance.ValuateFromString("100mm")

    'Get StartEndCutAngle
    Dim oStartEndCutAngle As Parameter
    Set oStartEndCutAngle = oObjStrFlange.StartEndCutAngle
    oStartEndCutAngle.ValuateFromString("60deg")

    'Get EndEndCutAngle
    Dim oEndEndCutAngle As Parameter
    Set oEndEndCutAngle = oObjStrFlange.EndEndCutAngle
    oEndEndCutAngle.ValuateFromString("60deg")

    'Set Edges
    Dim EdgeList(1) As Variant
    EdgeList(0) = 4
    EdgeList(1) = 3
    oObjStrFlange.SetEdges EdgeList

    'Get Edge
    Dim ObjSfdReferencePlane_3 As Reference
    Set ObjSfdReferencePlane_3 = oObjStrFlange.Edge
...
```

`Type`: Set and get the Type of conversion for flange. 1- Centered, 2- Tangent

`FlangeStartLimit`: Set and get the Flange Start Limit.

`FlangeEndLimit`: Set and get the Flange End Limit.

`WidthMeasurementType`: Set and get the Width Measurement Type. 1- FlangeWidthToInnerFace, 2- FlangeWidthToOuterFace, 3- FlangeWidthToNeutralFibre

`OperatedPlate`: Get the operated Plate.

`BendingAngle`: Get the Bending Angle and valuate it.

`BendingRadius`: Get the Bending Radius and valuate it.

`FlangeWidth`: Get the Flange Width and valuate it.

`StartEndCutOffset`: Get the Start EndCut Offset and valuate it.

`EndEndCutOffset`: Get the End EndCut Offset and valuate it.

`StartEndCutRadius`: Get the Start EndCut Radius and valuate it.

`EndEndCutRadius`: Get the End EndCut Radius and valuate it.

`StartEndCutDistance`: Get the Start EndCut Distance and valuate it.

`EndEndCutDistance`: Get the End EndCut Distance and valuate it.

`StartEndCutAngle`: Get the Start EndCut Angle and valuate it.

`EndEndCutAngle`: Get the End EndCut Angle and valuate it.

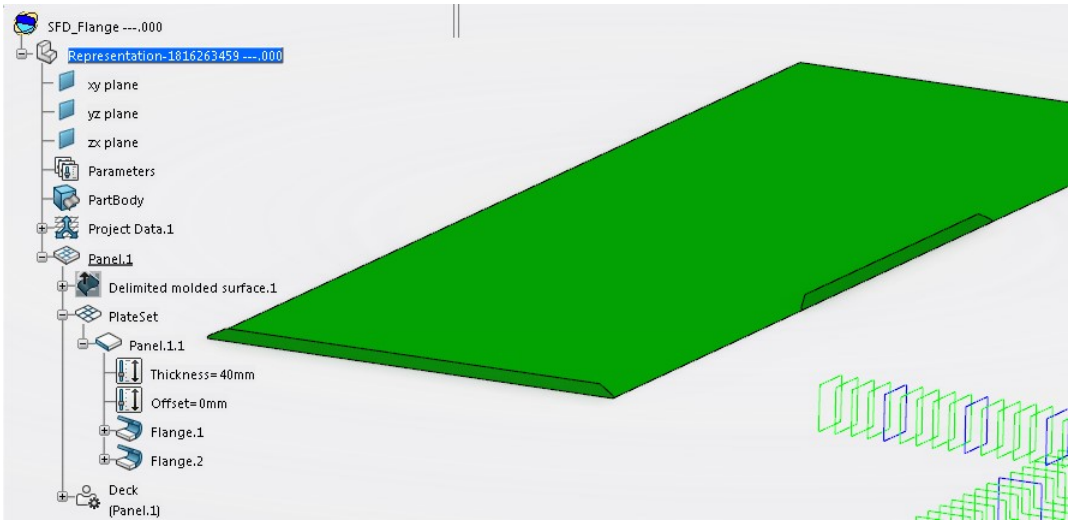
`SetEdges`: Set the Edges on which flanges to be created.

`Edge`: Get the Edge of Plate.

8. **Update the `Part` object**

```
...
    ObjPart.UpdateObject oObjStrFlange
End Sub
```

Fig.1: Created Flange




# Testing SFD Collar

This use case primarily focuses on the methodology to test collar.

Related Topics
Structure Functional Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcSteel_A90.3dxml`, `CAAScdSfdUcL8x8x1.3dxml`, `CAAScdSfdUcSlotSections.3dxml` `CAAScdSfdUcTestCollar.3dxml`, `CAAScdSfdUcCGR.3dxml` and `CAAScdSfdUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: CAAScdSfdUcTestCollarSource.htm

This use case can be divided in ten steps:

1. Searches and opens model which is named as "SFD_Collar"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves already created panel

1. **Searches and opens model which is named as "SFD_Collar"**

   As a first step, the UC retrieves a model "SFD_Collar" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_Collar"
 OpenProduct prdName , SFDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor is returned in **`SFDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **`SFDPrdEditor`** is used.

```
...
    Set SFDProdSel = SFDPrdEditor.Selection
...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves already created panel**

   In this step UC retrieves already created panel object. This panel is used as a penetrated element for the slot creation.

```
...
  Set RefObjSfdPanel = ObjPart.FindObjectByName("Longitudinal Bulkhead_2.1")
  Dim ObjSfdPanel As SfdPanel
  SFDProdSel.Add (RefObjSfdPanel)
  Set ObjSfdPanel = SFDProdSel.FindObject("CATIASfdPanel")
...
```

   `Find Object` method will find CATIASfdPanel panel and return it in `ObjSfdPanel` object.

5. **Retrieves sfd slot object**

   In this step UC finds a SFD slot. This slot is used for collar creation.

```
...
  Dim ObjStrSlot As StrSlot
  GetSlot ObjSfdPanel, ObjStrSlot
...

...
Sub GetSlot(iObjSfdPanel As SfdPanel, oObjStrSlot As StrSlot)

    Dim ObjStrSlots As StrSlots
    Set ObjStrSlots = iObjSfdPanel.StrSlots
    'Get StrSlot
    Set oObjStrSlot = ObjStrSlots.Item(1)

End Sub
...
```

   In above lines, `GetSlot` method finds slot object from given panel. This slot object is used for creation of collar.

6. **Retrieves sfd collar object**

   In this step UC finds a SFD collar on slot. This is the collar on which testing is to be done.

```
...
  Dim ObjStrCollar As StrCollar
  GetCollar ObjStrSlot, ObjStrCollar
...

...
Sub GetCollar(iObjStrSlot As StrSlot, oObjStrCollar As StrCollar)

   Dim ObjStrCollars As StrCollars
   Set ObjStrCollars = iObjStrSlot.Collars
   'Get StrCollar
   Set oObjStrCollar = ObjStrCollars.Item(1)

End Sub
...
```

   In above lines, `GetCollar` method finds collar object from given slot. This is the collar used for testing.

7. **Setting up collar parameters**

   In this step UC sets up collar parameters.

```
...
  SetCollarParameters ObjStrCollar
...

...
Sub SetCollarParameters(iObjStrCollar As StrCollar)

   iObjStrCollar.Material = "Steel A90"

   iObjStrCollar.MaterialThrowOrientation = 1

   Dim oThickness As Parameter
   Set oThickness = iObjStrCollar.Thickness
   oThickness.ValuateFromString("20mm")

   Dim oMaterialThrowOrientation As CATStrCollarThrowOrientation
   oMaterialThrowOrientation = iObjStrCollar.MaterialThrowOrientation

   Dim oMaterial As String
   oMaterial = iObjStrCollar.Material

   Dim ObjCollarParameters As StrParameters
   Set ObjCollarParameters = iObjStrCollar.CollarParameters
   Dim Width As Parameter
   Set Width = ObjCollarParameters.Item(1)
   Width.ValuateFromString("205mm")
   Dim Angle As Parameter
   Set Angle = ObjCollarParameters.Item(2)
   Angle.ValuateFromString("35deg")

   iObjStrCollar.Update

End Sub
...
```

In above lines, `SetCollarParameters` method sets up different properties on collar. Here, `Material` property is used to get and set collar material. `Thickness` is used to get collar thickness and then valuate it as required. `MaterialThrowOrientation` is used to set and get different orientations of collar material (0 - catStrCollarThrowOrientationInvert, 1 - catStrCollarThrowOrientationNormal, 2 - catStrCollarThrowOrientationCentered). `CollarParameters` will get the public parameters on particular collar. Valuate it as required. `Update` updates the modified collar object.

8. **Adding Collar on Slot**

   In this step UC adds a SFD collar on slot.

```
...
  Dim ObjStrCollar_2 As StrCollar
  AddCollar  ObjStrSlot, ObjStrCollar_2
...

...
Sub AddCollar(iObjStrSlot As StrSlot, oObjStrCollar_2 As StrCollar)

   Dim ObjStrCollars As StrCollars
   Set ObjStrCollars = iObjStrSlot.Collars

   'Add StrCollar
   Set oObjStrCollar_2 = ObjStrCollars.Add

End Sub
...
```

   In above lines, `AddCollar` method creates new collar object on given slot.

9. **Removing Collar on Slot**

   In this step UC removes a SFD collar on slot.

```
...
  RemoveCollar ObjStrSlot
...

...
Sub RemoveCollar(iObjStrSlot As StrSlot)

    Dim ListOfCollars As StrCollars
    Set ListOfCollars = iObjStrSlot.Collars

    'Remove the newly added collar
    Dim ObjStrCollar As StrCollar
    Set ObjStrCollar = ListOfCollars.Item(2)
    ListOfCollars.Remove (ObjStrCollar)

End Sub
...
```

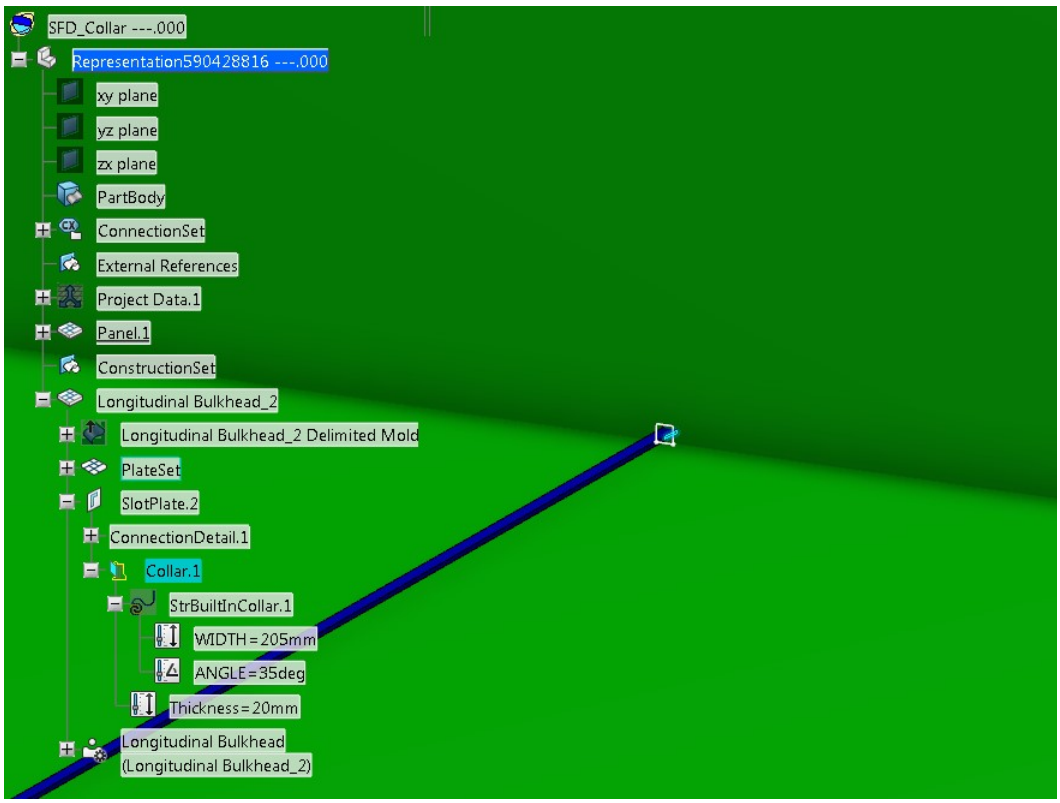
   In above lines, `RemoveCollar` method removes newly created collar object on given slot.

10. **Updates the Part object**

```
...
    ObjPart.Update
End Sub
```

   `Update` method updates the `ObjPart`.

Fig.1: Tested Collar




# Creating an SFD Opening On Profile Using a Sketch Profile

This use case primarily focuses on the methodology to create a SFD Opening on profile using sketch profile.

Related Topics
Structure Functional Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcCGR.3dxml, CAAScdSfdUcSR.3dxml, CAAScdSfdUcSteel_A90.3dxml,CAAScdSfdUcWT18x179_5.3dxm, CAAScdSfdUcPart.3dxml, CAAScdSfdUcCreateOpeningOnProfile.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Where to find the macro: CAAScdSfdUcCreateOpeningOnProfileSketchSource.htm

This use case can be divided in nine steps:

1. Searches and opens model which is named as "SFD_CreateOpeningOnProfile"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves a SFD stiffener object
5. Retrieves first sketch profile
6. Creates opening on profile with limit mode set to UpToLast
7. Retrieves second sketch profile
8. Creates opening on profile with limit mode set to dimensions(Limit1/Limit2)
9. Updates the Part object

1. **Searches and opens model which is named as "SFD_CreateOpeningOnProfile"**

   As a first step, the UC retrieves a model "SFD_CreateOpeningOnProfile" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_CreateOpeningOnProfile"
 OpenProduct prdName , SFDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor i
   **`SFDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **`SFDPrdEditor`** is used.

```
 ...
    Set SFDProdSel = SFDPrdEditor.Selection
 ...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves a SFD panel object**

   In this step UC retrieves stiffener object.

```
...
  Set RefObjSfdStiffener = ObjPart.FindObjectByName("Stiffener.1")
  Dim ObjSfdStiffener As StrSfdStiffener
  SFDProdSel.Add RefObjSfdStiffener
  Set ObjSfdStiffener = SFDProdSel.FindObject("CATIASfdStiffener")
...
```

   `FindObjectByName` method finds object whose name is "Stiffener.1" and returns reference to it. Here RefObjSfdStiffener is of type `Reference`. To retrieve `StrSf` `Reference` add retrieved reference to the selection and call `FindObject` method as shown above. This will return the `StrSfdStiffener` object variable.

5. **Retrieves first sketch profile**

   In this step UC retrieves a sketch profile.

```
...
  Set ProfileSketchOpeningUpToLast = ObjPart.FindObjectByName("Profile.2")
  Set RefProfileSketchOpeningUpToLast = ObjPart.CreateReferenceFromObject(ProfileSketchOpeningUpToLast)
...
```

   `FindObjectByName` method finds object whose name is "Profile.2" and returns reference to it. Here RefProfileSketchOpeningUpToLast is of type `Reference`. To `RefProfileSketchOpeningUpToLast` object from `ProfileSketchOpeningUpToLast` object `CreateReferenceFromObject` method is used.

6. **Creates opening with limit mode set to UpToLast**

   Now, panel and sketch profile are available to create opening. Call `CreateOpeningOnProfileSketchUpToLast` method to create opening on profile using sketch `CreateOpeningOnProfileSketchUpToLast` method takes a profile and a sketch profile as input parameters and it returns created opening on it as output paramet `ObjStrOpeningOnProfileWtLtMdUpToLast`.

```
...
  Dim ObjStrOpeningOnProfileWtLtMdUpToLast As StrOpeningOnProfile
  CreateOpeningOnProfileSketchUpToLast ObjSfdStiffener, RefProfileSketchOpeningUpToLast, ObjStrOpeningOnProfileWtLtMdUpToLast
...
```

```
...
  Sub CreateOpeningSketchUpToLast(iObjSfdStiffener As StrSfdStiffener, RefProfileSketch As Reference, oObjStrOpeningOnProfile As S

    'Add Opening On Profile
    AddOpening iObjSfdStiffener, oObjStrOpeningOnProfile

    'Get StrOpening from StrOpeningOnProfile
    Dim ObjStrOpening As StrOpening
    Set ObjStrOpening = oObjStrOpeningOnProfile.StrOpening

    'Set opening type catStrOpeningModeOutputProfile for Sketch opening
    ObjStrOpening.OpeningType = catStrOpeningModeOutputProfile

    'Set category
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = ObjStrOpening.StrCategoryMngt
    ObjStrCategoryMngt.SetCategory "SldOpening"

    'Set forming mode
    'ObjStrOpening.FormingMode = 0
    Set ObjStrOpeningExtrusionMngt = ObjStrOpening.StrOpeningExtrusionMngt
    ObjStrOpeningExtrusionMngt.ExtrusionMode = 0

    'Set output profile
    'Set LimitMode as 0 for UpTOLast
    'Get object of StrOpeningOutputProfile
    Dim ObjStrOpeningOutputProfile As StrOpeningOutputProfile
    Set ObjStrOpeningOutputProfile = ObjStrOpening.StrOpeningOutputProfile
    'Set OutputProfile for sketch opening
    ObjStrOpeningOutputProfile.OutputProfile = RefProfileSketch
    'Set direction
    Set ObjRefDirection = Manager.GetReferencePlane(ObjPart, 2, "CROSS.30")
    Set RefDirection = ObjPart.CreateReferenceFromObject(ObjRefDirection)
    ObjStrOpeningOutputProfile.Direction = RefDirection
    'Set LimitMode to UpToLast
    ObjStrOpeningOutputProfile.LimitMode = 0

    ObjPart.UpdateObject oObjStrOpeningOnProfile
    ObjPart.UpdateObject iObjSfdStiffener

  End Sub
...
```

   Set various required parameters to get the desired opening on profile with UpToLast limit mode.

7. **Retrieves second sketch profile**

```
...
  Set ProfileSketchOpeningDimensions = ObjPart.FindObjectByName("Profile.3")
  Set RefProfileSketchOpeningDimensions = ObjPart.CreateReferenceFromObject(ProfileSketchOpeningDimensions)
...
```

   `FindObjectByName` method finds object whose name is "Profile.3" and returns reference to it. Here `RefProfileSketchOpeningDimensions` is of type `Referenc` `RefProfileSketchOpeningDimensions` object from `ProfileSketchOpeningDimensions` object `CreateReferenceFromObject` method is used.

8. **Creates opening on profile with limit mode set to dimensions(Limit1/Limit2)**

   Call `CreateOpeningOnProfileSketchDimensions` method to create opening on profile using sketch profile. `CreateOpeningOnProfileSketchDimensions` met

sketch profile as input parameters and it returns created opening as output parameter in `ObjStrOpeningOnProfileWtLtMdDim`.

```
...
  Dim ObjStrOpeningOnProfileWtLtMdDim As StrOpeningOnProfile
  CreateOpeningOnProfileSketchDimensions ObjSfdStiffener, RefProfileSketchOpeningDimensions, ObjStrOpeningOnProfileWtLtMdDim
...

...
  Sub CreateOpeningSketchDimensions(iObjSfdStiffener As StrSfdStiffener, RefProfileSketch As Reference, oObjStrOpeningOnProfile As

    'Add Opening On Profile
    AddOpening iObjSfdStiffener, oObjStrOpeningOnProfile

    'Get StrOpening from StrOpeningOnProfile
    Dim ObjStrOpening As StrOpening
    Set ObjStrOpening = oObjStrOpeningOnProfile.StrOpening

    'Set opening type catStrOpeningModeOutputProfile for Sketch opening
    ObjStrOpening.OpeningType = catStrOpeningModeOutputProfile

    'Set category
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = ObjStrOpening.StrCategoryMngt
    ObjStrCategoryMngt.SetCategory "SldOpening"

    'Set forming mode
    'ObjStrOpening.FormingMode = 0
    Set ObjStrOpeningExtrusionMngt = ObjStrOpening.StrOpeningExtrusionMngt
    ObjStrOpeningExtrusionMngt.ExtrusionMode = 0

    'Set output profile
    'Set LimitMode to 1 for Dimensions
    'Get object of StrOpeningOutputProfile
    Dim ObjStrOpeningOutputProfile As StrOpeningOutputProfile
    Set ObjStrOpeningOutputProfile = ObjStrOpening.StrOpeningOutputProfile
    'Set OutputProfile for sketch opening
    ObjStrOpeningOutputProfile.OutputProfile = RefProfileSketch
    'Set direction
    Set ObjRefDirection = Manager.GetReferencePlane(ObjPart, 2, "CROSS.30")
    Set RefDirection = ObjPart.CreateReferenceFromObject(ObjRefDirection)
    ObjStrOpeningOutputProfile.Direction = RefDirection
    'Set LimitMode to UpToLast
    ObjStrOpeningOutputProfile.LimitMode = 1

    'Get dimensionsMngt object to set offsets
    Dim ObjStrOpeningLimitDimensionsMngt As StrOpeningLimitDimensionsMngt
    Set ObjStrOpeningLimitDimensionsMngt = ObjStrOpening.StrOpeningLimitDimensionsMngt

    'Set FirstOffset of opening output profile
    Dim FirstOffset As Parameter
    Set FirstOffset = ObjStrOpeningLimitDimensionsMngt.GetFirstOffset
    FirstOffset.ValuateFromString ("1000mm")
    'Set second Offset of opening output profile
    Dim SecondOffsetOffset As Parameter
    Set SecondOffsetOffset = ObjStrOpeningLimitDimensionsMngt.GetSecondOffset
    SecondOffsetOffset.ValuateFromString ("-2000mm")

    'Invert the first offset and second offset
    ObjStrOpeningLimitDimensionsMngt.Invert

    'Update the opening
    ObjPart.UpdateObject oObjStrOpeningOnProfile
    ObjPart.UpdateObject iObjSfdStiffener

  End Sub
...
```

Set various required parameters to get the desired opening on profile with Dimensions limit mode.

9. **Updates the Part object**

```
...
    ObjPart.Update
End Sub
```

`Update` method updates the `ObjPart`

**Detailed steps of method called in the use case**

- **AddOpening method**

```
Sub AddOpening(iObjSfdStiffener As StrSfdStiffener, oObjStrOpeningOnProfile As StrOpeningOnProfile)
    'Get StrOpeningsOnProfile object
    Dim ObjStrOpeningsOnProfile As StrOpeningsOnProfile
    Set ObjStrOpeningsOnProfile = iObjSfdStiffener.StrOpeningsOnProfile
    'Add opening
    Set oObjStrOpeningOnProfile = ObjStrOpeningsOnProfile.Add
End Sub
```

Method `AddOpening`takes a stiffener object `iObjSfdStiffener` as input parameter and it returns created opening as output parameter in `oObjStrOpeningOnProf`. of type `StrOpeningsOnProfile` is retrieved in `ObjStrOpeningsOnProfile`. Then on this object `Add` method is called to create an opening with no properties set.

# Creating a Standard Opening On Profile

This use case primarily focuses on the methodology to create Standard Opening On Profile.

Related Topics

Structure Functional Design Object

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcCGR.3dxml`, `CAAScdSfdUcSR.3dxml`,`CAAScdSfdUcSteel_A90.3dxml`,

`CAAScdSfdUcWT18x179_5.3dxml, CAAScdSfdUcPart.3dxml, CAAScdSfdUcCreateOpeningOnProfile.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Model Map
Launching an Automation Use Case

Where to find the macro: CAAScdSfdUcCreateOpeningOnProfileStandardSource.htm

This use case can be divided in six steps:

1. Searches and opens model which is named as "SFD_CreateOpeningOnProfile"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves service of type RfgService
5. Retrieves a SFD stiffener object
6. Create standard Opening On Profile using Offset/Offset positioning strategy

1. **Searches and opens model which is named as "SFD_CreateOpeningOnProfile"**

   As a first step, the UC retrieves a model "SFD_CreateOpeningOnProfile" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_CreateOpeningOnProfile"
 OpenProduct prdName , SFDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor i returned in **`SFDPrdEditor`**.

2. **Retrieves Selection object**

   In this step UC retrieves Selection object in `SFDProdSel` variable.

```
...
  Set SFDProdSel = SFDPrdEditor.Selection
...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object in `ObjPart` variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves service of type RfgService**

   In this step UC retrieves `RfgService`.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

   `GetService` method returns `RfgService`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

5. **Retrieves a SFD stiffener object**

   In this step UC retrieves stiffener object.

```
...
  Set RefObjSfdStiffener = ObjPart.FindObjectByName("Stiffener.1")
  Dim ObjSfdStiffener As StrSfdStiffener
  SFDProdSel.Add RefObjSfdStiffener
  Set ObjSfdStiffener = SFDProdSel.FindObject("CATIASfdStiffener")
...
```

   `FindObjectByName` method finds object whose name is "Stiffener.1" and returns reference to it. Here RefObjSfdStiffener is of type `Reference`. To retrieve `StrSfdStiffener` object from `Reference` add retrieved reference to the selection and call `FindObject` method as shown above. This will return the `StrSfdStiffener` object variable.

6. **Create Standard Opening On Profile with Offset/Offset positioning strategy**

   Call `CreateStandardOpeningOnProfileOffsetOffset` method to create a standard opening on profile with offset/offset positioning strategy. `CreateStandardOpeningOnProfileOffsetOffset` method takes a stiffener object as input parameter and returns created opening as output parameter in `ObjStrOpeningOnProfile`.

```
...
  Dim ObjStrOpeningOnProfile As StrOpeningOnProfile
  CreateStandardOpeningOnProfileOffsetOffset ObjSfdStiffener, ObjStrOpeningOnProfile
  ObjPart.Update
...
```

   The method `CreateStandardOpeningOnProfileOffsetOffset` is detailed as in the below sub steps.

```
...
Sub CreateStandardOpeningOnProfileOffsetOffset(iObjSfdStiffener As StrSfdStiffener, oObjStrOpeningOnProfile As StrOpeningOnProfile

    'Add Opening On Profile
    AddOpening iObjSfdStiffener, oObjStrOpeningOnProfile

    'Get StrOpening from StrOpeningOnProfile
    Dim ObjStrOpening As StrOpening
    Set ObjStrOpening = oObjStrOpeningOnProfile.StrOpening

    'Set opening type catStrOpeningModeStandard for standard opening
    ObjStrOpening.OpeningType = catStrOpeningModeStandard
```

```
    'Set category
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = ObjStrOpening.StrCategoryMngt
    ObjStrCategoryMngt.SetCategory "ManHole"

    'Get StrOpeningStandard from StrOpening
    Dim ObjStrOpeningStandard As StrOpeningStandard
    Set ObjStrOpeningStandard = ObjStrOpening.StrOpeningStandard

    'Set standard mode type
    ObjStrOpeningStandard.StandardModeType = catStrOpeningSTDRectMode

    'Get contour and set parameters data
    Dim ContourName As String
    Dim StdContourParms As StrStandardContourParameters
    'Get contour names
    Dim ContourNames() As Variant
    Dim ObjStrOpeningsMgr As StrOpeningsMgr
    Set ObjStrOpeningsMgr = iObjSfdStiffener.StrOpeningsMgr
    'Get available contour names
    ObjStrOpeningsMgr.GetAvailableStandardContours ContourNames
    ContourName = ContourNames(1)
   'Get parameters of contour
    Set StdContourParms = ObjStrOpeningsMgr.GetStandardContourParms(ContourName)
    'Set contour parameters data
    SetRectContourParamsData StdContourParms, "100mm", "50mm", "5mm"

    'Get positioning strategy and set parameters data
    Dim PosStratName As String
    Dim PosStratParms As StrStandardPosStrategyParameters
    'Get std positiong strategy names
    Dim StdPosStrategyNames() As Variant
    'Get available standard positioning strategy names
    ObjStrOpeningsMgr.GetAvailableStandardPositioningStrategiesForProfile StdPosStrategyNames
    'Select positiong strategy offset offset
    PosStratName = StdPosStrategyNames(2)
    'Get standard positioining strategy parameters
    Set PosStratParms = ObjStrOpeningsMgr.GetStandardPositioningStrategyParmsForProfile(PosStratName)

    'Set standard opening remaining parameters
    'Set direction
    ObjStrOpeningStandard.DirectionForOpeningOnProfile = FALSE
    'Set extrusion mode
    'Set ObjStrOpeningExtrusionMngt = ObjStrOpening.StrOpeningExtrusionMngt
    'ObjStrOpeningExtrusionMngt.ExtrusionMode = 0
    'Set limit mode
    ObjStrOpeningStandard.LimitMode = 0
    'Set contour and position strategy for profile
    ObjStrOpeningStandard.SetContourAndPosStrategyForProfile ContourName, StdContourParms, PosStratName, PosStratParms

    'Get Reference of profile
    Set RefObjSfdStiffener = ObjPart.FindObjectByName("Stiffener.1")

    'set Offset Offset Position Strategy Parameters
    Set ObjRefElem = Manager.GetReferencePlane(ObjPart, 3, "LONG.14")
    Set RefElem = ObjPart.CreateReferenceFromObject(ObjRefElem)
    ObjStrOpeningStandard.SetOffsetOffsetPosStratParms RefObjSfdStiffener, RefElem, "100mm", "100mm", "Gravity", "40deg"

    'Update created opening on profile object
    ObjPart.UpdateObject oObjStrOpeningOnProfile
    ObjPart.UpdateObject iObjSfdStiffener

End Sub
...
```

`DirectionForOpeningOnProfile`: Sets the direction for opening on profile.

`StandardModeType`: Sets the mode for standard opening on profile. 0 -> catStrOpeningSTDUndefinedMode, 1 -> catStrOpeningSTDRoundMode, 2 -> catStrOpeningSTDRectMode, 3 -> catStrOpeningSTDOblongMode, 4 -> catStrOpeningSTDCatalogMode

`SetContourAndPosStrategyForProfile`: Sets the contour and position strategy for opening on profile.

`SetOffsetOffsetPosStratParms`: There are in all 6 parameters for this method. 1. The reference of the profile, 2.The reference element used, 3. The horizontal offset distance from reference element, 4. The vertical offset distance from anchor point, 5. The anchor point type, 6. The axis angle.

Simillarly, for **Mid Dist/Offset** and **Spacing/Offset** Position Strategy cases `SetMidDistOffsetPosStratParms` and `SetSpacingOffsetPosStratParms` methods are called respectively.

`SetMidDistOffsetPosStratParms`: There are in all 6 parameters for this method. 1. The reference of the profile, 2.The first reference element used, 3. The second reference element used, 4. The vertical offset distance from anchor point, 5. The anchor point type, 6. The axis angle.

`SetSpacingOffsetPosStratParms`: There are in all 7 parameters for this method. 1. The reference of the profile, 2. Whether reference point is from start or end, 3. The horizontal offset distance, 4. Whether mode is absolute or relative, 5. The vertical offset distance from anchor point, 6. The anchor point type, 7. The axis angle.

The source also includes standard opening on profile creation using positioning strategies Mid Dist/Offset, Spacing/Offset. To see the source go to CAAScdSfdUcCreateOpeningOnProfileStandardSource.htm.

# Creating an SFD Opening On Profile Using 3D Objects

This use case primarily focuses on the methodology to create a SFD Opening on profile using 3D object.

Related Topics
Structure Functional Design Object Model Map

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSfdUcCGR.3dxml`, `CAAScdSfdUcSR.3dxml`, `CAAScdSfdUcWT18x179_5.3dxml`,`CAAScdSfdUcSteel_A90.3dxml`, `CAAScdSfdUcPart.3dxml`, `CAAScdSfdUcCreateOpeningOnProfile.3dxml` files supplied in folder

`InstallRootFolder\CAADoc\Doc\English\CAAScdSfdSFDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Launching an Automation Use Case

Where to find the macro: CAAScdSfdUcCreateOpeningOnProfile3DObjectSource.htm

This use case can be divided in seven steps:

1. Searches and opens model which is named as "SFD_CreateOpeningOnProfile"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves a SFD stiffener object
5. Get 3D Object for creating opening on profile
6. Creates opening on profile
7. Removes opening on profile
8. Updates the Part object

1. **Searches and opens model which is named as "SFD_CreateOpeningOnProfile"**

   As a first step, the UC retrieves a model "SFD_CreateOpeningOnProfile" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SFD_CreateOpeningOnProfile"
 OpenProduct prdName , SFDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SFD model from underlying database the current active editor i returned in **`SFDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **`SFDPrdEditor`** is used.

```
 ...
  Set SFDProdSel = SFDPrdEditor.Selection
 ...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set ObjPart = SFDPrdEditor.ActiveObject
...
```

4. **Retrieves a SFD stiffener object**

   In this step UC retrieves stiffener object.

```
...
  Set RefObjSfdStiffener = ObjPart.FindObjectByName("Stiffener.1")
  Dim ObjSfdStiffener As StrSfdStiffener
  SFDProdSel.Add RefObjSfdStiffener
  Set ObjSfdStiffener = SFDProdSel.FindObject("CATIASfdStiffener")
...
```

   `FindObjectByName` method finds object whose name is "Stiffener.1" and returns reference to it. Here RefObjSfdStiffener is of type `Reference`. To retrieve `StrSfdStiffener` object from `Reference` add retrieved reference to the selection and call `FindObject` method as shown above. This will return the `StrSfdStif` object variable.

5. **Get 3D Object for creating opening on profile**

```
...
    Set Obj3DOpeningOnProfileDimensions = StiffenerPart.FindObjectByName("Extrude.1")
    Dim Ref3DObjectOpeningOnProfileDimensions As Reference
    Set Ref3DObjectOpeningOnProfileDimensions = ObjPart.CreateReferenceFromObject(Obj3DOpeningOnProfileDimensions)
...
```

6. **Creates opening on profile**

   Now, stiffener is available to create opening on it. Call `CreateOpeningOnProfile3DObject` method to create opening on profile. `CreateOpeningOnProfile3DO` method takes a profile as input parameter and it returns created opening as output parameter in `ObjStrOpeningOnProfile`.

```
...
  Dim ObjStrOpeningOnProfile As StrOpeningOnProfile
  CreateOpeningOnProfile3DObject ObjSfdStiffener, Ref3DObjectOpeningOnProfileDimensions, ObjStrOpeningOnProfile
...

...
  Sub CreateOpening3DObject(iObjSfdStiffener As StrSfdStiffener, Ref3DObject As Reference, oObjStrOpeningOnProfile As StrOpeningOn

    'Add Opening On Profile
    AddOpening iObjSfdStiffener, oObjStrOpeningOnProfile

    'Get StrOpening from StrOpeningOnProfile
    Dim ObjStrOpening As StrOpening
    Set ObjStrOpening = oObjStrOpeningOnProfile.StrOpening

    'Set opening type catStrOpeningMode3DObject for 3dobject opening
    ObjStrOpening.OpeningType = catStrOpeningMode3DObject

    'Set category
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = ObjStrOpening.StrCategoryMngt
    ObjStrCategoryMngt.SetCategory "SldOpening"
```

```
    'Set intersecting element for opening creation
    Dim ObjStrOpening3DObject As StrOpening3DObject
    Set ObjStrOpening3DObject = ObjStrOpening.StrOpening3DObject
    ObjStrOpening3DObject.IntersectingElement = Ref3DObject

    'Set forming mode
    Set ObjStrOpeningExtrusionMngt = ObjStrOpening.StrOpeningExtrusionMngt
    ObjStrOpeningExtrusionMngt.ExtrusionMode = 0
    'ObjStrOpening.FormingMode = 0

    'Update the created opening
    ObjPart.UpdateObject oObjStrOpeningOnProfile
    ObjPart.UpdateObject iObjSfdStiffener

  End Sub
...

...
Sub AddOpening(iObjSfdStiffener As StrSfdStiffener, oObjSfdOpeningOnProfile As StrOpeningOnProfile)
    Dim ObjStrOpeningsOnProfile As StrOpeningsOnProfile
    Set ObjStrOpeningsOnProfile = iObjSfdStiffener.GetOpeningsOnProfile(0)
    'Add opening
    Set oObjSfdOpeningOnProfile = ObjStrOpeningsOnProfile.Add

End Sub
...
```

7. **Removes opening on profile**

```
...
    Dim ObjStrOpeningsOnProfile As StrOpeningsOnProfile
    Set ObjStrOpeningsOnProfile = ObjSfdStiffener.GetOpeningsOnProfile(0)
    ObjStrOpeningsOnProfile.Remove ObjStrOpeningOnProfile
...
```

Object of `StrOpeningsOnProfile` is retrieved in `ObjStrOpeningsOnProfile`. Then on `ObjStrOpeningsOnProfile` object `Remove` method is called to remove th opening. This method takes a input parameter opening object to remove.

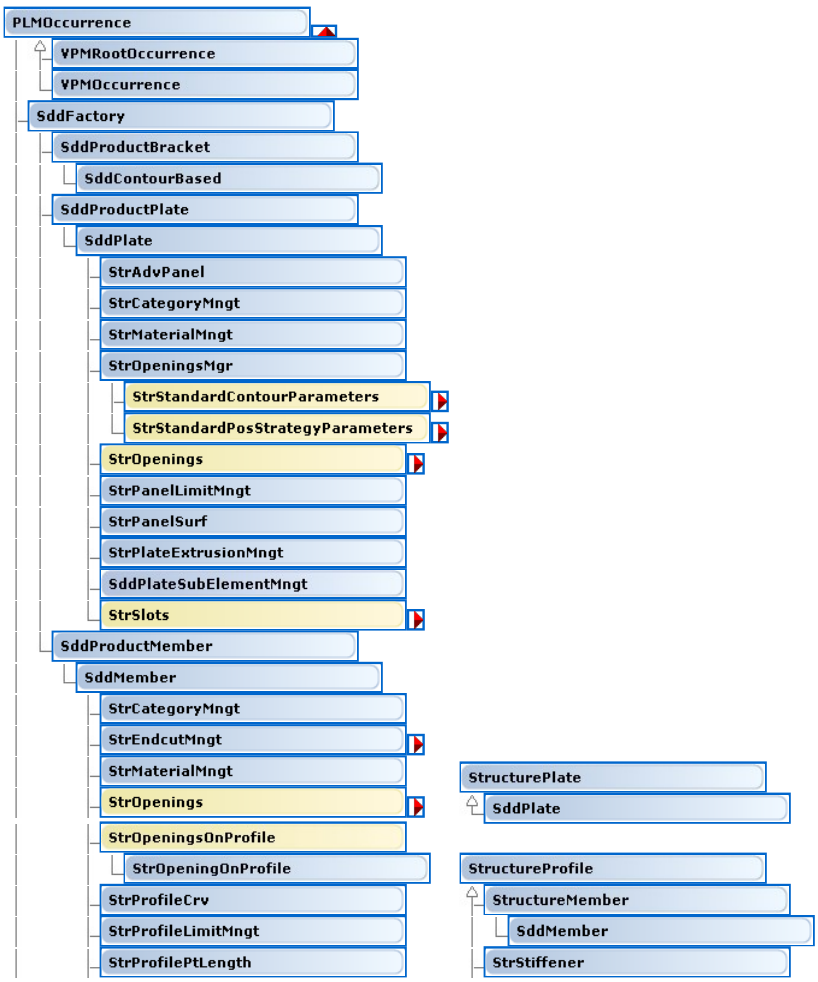
8. **Updates the Part object**

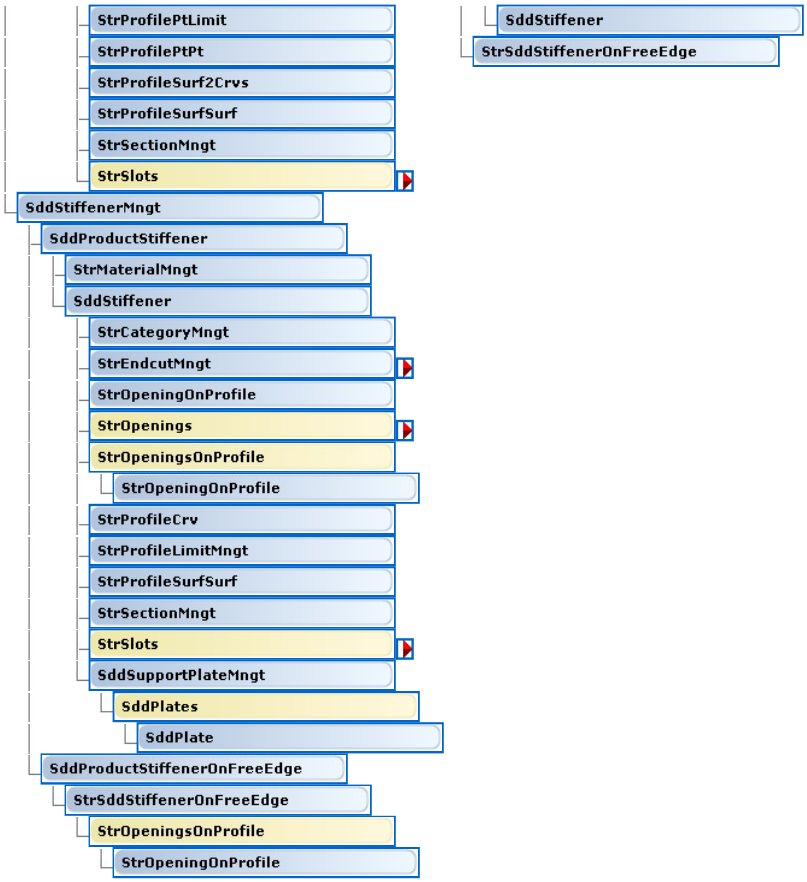
```
...
    ObjPart.Update
End Sub
```

`Update` method updates the `ObjPart`

# Structure Design Object Model Map

See Also Legend






# Creating an SDD Plate

This use case primarily focuses on the methodology to create SDD plate.

Related Topics
Structure Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcCGR.3dxml, CAAScdSddUcSR.3dxml, CAAScdSddUcSteel_A42.3dxml, CAAScdSddUcCreatePlate.3dxml` files supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcCreatePlateSource.htm

This use case can be divided in five steps:

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Part object
3. Retrieves Service manager (RfgService)
4. Retrieves SddFactory for creating SDD objects
5. Creates a Plate

1. **Searches and opens model which is named as "SddProduct"**

   As a first step, the use case retrieves a model "SddProduct" from the database, loads it, and returns object of the Editor.

```
...
 Dim SDDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SDDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SDDPrdEditor`**, an Editor object. After searching and opening the SFD model from the underlying database, the current active editor is returned in **`SDDPrdEditor`**.

2. **Retrieves Part object**

   In this step, the use case retrieves Part object ObjPart variable.

```
...
  Dim product1Service  As  PLMProductService
  Set product1Service = DDPrdEditor.GetService("PLMProductService")
  Dim ObjVPMRootOccurrence  As  VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference  As  VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances  As  VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjPart = ObjVPMRepReference.GetItem("Part")
```

```
...
```

3. **Retrieves Service manager (RfgService)**

   In this step, the use case retrieves `RfgService`.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

   `GetService` method returns `RfgService`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

4. **Retrieves SddFactory for creating SDD objects**

   In this step, the use case retrieves the SddFactory for creating SDD objects.

```
...
  Dim ObjSddFactory  As  SddFactory
  SDDProdSel.Add ObjVPMRootOccurrence
  Set ObjSddFactory = SDDProdSel.FindObject("CATIASddFactory")
...
```

5. **Creates a plate**

   Call `CreatePlate` method to create a plate. `CreatePlate` method takes the SddFactory as input parameter and it returns the SddProductPlate as output parameter in `ObjSddProductPlate`.

```
...
  Dim ObjSddProductPlate As SddProductPlate
  CreatePlate ObjSddFactory, ObjSddProductPlate
...
```

   The method `CreatePlate` is detailed as in the below sub steps.

   i. **Creates plate with no properties then set the properties category, support, material and thickness**

```
Sub CreatePlate(iObjSddFactory As SddFactory, oObjSddProductPlate As SddProductPlate)

    'Create plate with no properties set
    Set oObjSddProductPlate = iObjSddFactory.AddProductPlate(False)
    Dim ObjSddPlate As SddPlate
    Set ObjSddPlate = = oObjSddProductPlate.SddPlate

    'Set category
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = ObjSddPlate.StrCategoryMngt
    ObjStrCategoryMngt.AutomaticName = True
    ObjStrCategoryMngt.SetCategory "DeckPanel"

    'Set panel support
    Set ObjPlateSupport = Manager.GetReferencePlane(ObjPart, 1, "DECK.1")
    Set PlateSupport = ObjPart.CreateReferenceFromObject(ObjPlateSupport)
    Dim ObjStrPanelSurf As StrPanelSurf
    Set ObjStrPanelSurf = ObjSddPlate.StrPanelSurf
    ObjStrPanelSurf.Support = PlateSupport
    'Set material
    Dim ObjMaterialMngt As StrMaterialMngt
    Set ObjMaterialMngt = oObjSddProductPlate.StrMaterialMngt
    ObjMaterialMngt.SetMaterial "Steel A42"

    'Set thickness and throw orientation
    Dim ObjStrPlateExtrusionMngt As StrPlateExtrusionMngt
    Set ObjStrPlateExtrusionMngt = ObjSddPlate.StrPlateExtrusionMngt
    ObjStrPlateExtrusionMngt.ThrowOrientation = 0
    ObjStrPlateExtrusionMngt.OffsetMode = 0
    Dim Thickness As Parameter
    Set Thickness = ObjStrPlateExtrusionMngt.GetThickness
    Thickness.ValuateFromString "10mm"

    'Set panel limits
    SetPanelLimits ObjSddPlate
...
```

   In this method first `AddProductPlate` method is called on iObjSddFactory object to create a new empty panel and stores it in `oObjSddProductPlate` which is a output parameter of this method. Next, it retrieves the object of type `StrCategoryMngt` in `ObjStrCategoryMngt` and calls `SetCategory` method to set the category. Similarly for setting the panel support, it retrieves the object of type `StrPanelSurf` in `ObjStrPanelSurf` and sets `Support` property with the reference to "Deck.1". For setting the material, it retrieves the object of type `StrMaterialMngt` in `ObjStrMaterialMngt` and calls `SetMaterial` method to set the material. Now to set the thickness and throw orientation, it retrieves the object of type `StrPlateExtrusionMngt` in `ObjStrPlateExtrusionMngt` and sets `ThrowOrientation`, `OffsetMode` properties. Finally `SetPanelLimits` method is called to set the limits of the panel. The method `SetPanelLimits`is detailed in the below steps. The method `SetPanelLimits` takes SddPlate as input parameter and sets the limits on this plate

   ii. **Sets panel limits**

```
Sub SetPanelLimits(iObjSddPlate As SddPlate)

'   1- Retrieves StrPanelLimitMngt for setting limits
    Set oObjProdMember = iObjSddFactory.AddProductMember
    Dim ObjStrPanelLimitMngt As StrPanelLimitMngt
    Set ObjStrPanelLimitMngt = iObjSddPlate.StrPanelLimitMngt

'   2- Sets limits on the panel
    Set ObjLimit1 = Manager.GetReferencePlane(ObjPart, 2, "CROSS.60")
    Set Limit1 = ObjPart.CreateReferenceFromObject(ObjLimit1)
    ObjStrPanelLimitMngt.SetLimitingObject Limit1, -1, 0, 2

    Set ObjLimit2 = Manager.GetReferencePlane(ObjPart, 2, "CROSS.120")
    Set Limit2 = ObjPart.CreateReferenceFromObject(ObjLimit2)
    ObjStrPanelLimitMngt.SetLimitingObject Limit2, -1, -1, 2
```

```
    Set ObjLimit3 = Manager.GetReferencePlane(ObjPart, 3, "LONG.-18")
    Set Limit3 = ObjPart.CreateReferenceFromObject(ObjLimit3)
    ObjStrPanelLimitMngt.SetLimitingObject Limit3, -1, 0, 2

    Set ObjLimit4 = Manager.GetReferencePlane(ObjPart, 3, "LONG.18")
    Set Limit4 = ObjPart.CreateReferenceFromObject(ObjLimit4)
    ObjStrPanelLimitMngt.SetLimitingObject Limit4, -1, -1, 2


End Sub
```

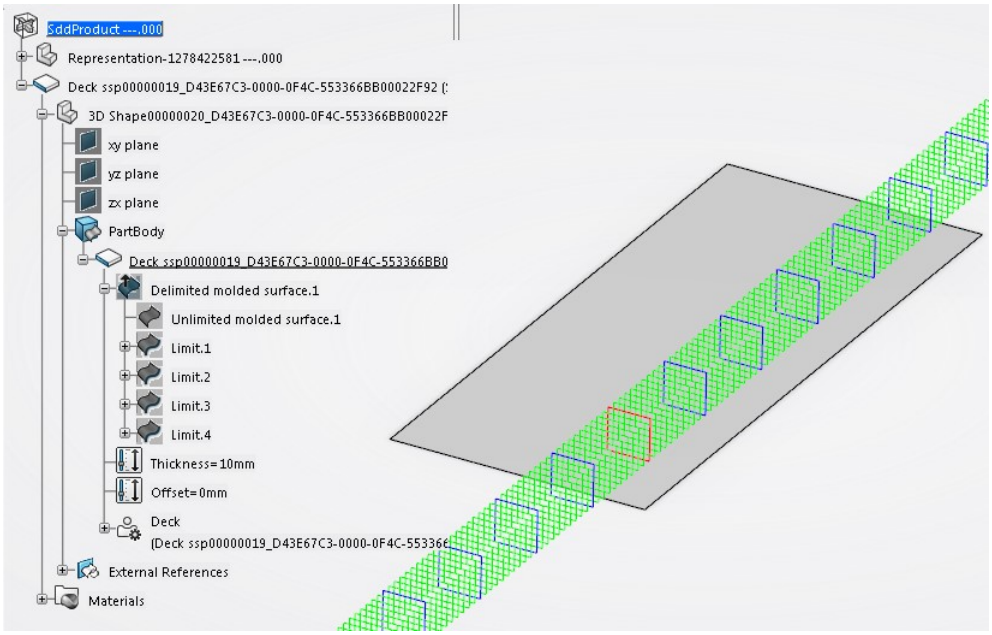
As a first step, this method retrieves the object of type `StrPanelLimitMngt` in `ObjStrPanelLimitMngt`. Then references to the limiting objects are created using `CreateReferenceFromObject` method which are references to the planes "CROSS.60", "CROSS.120", "LONG.-18" and "LONG.18" obtained by `GetReferencePlane` method of RfgService. These reference planes are then used as limiting objects for the panel using `SetLimitingObject`method of `StrPanelLimitMngt`

iii. **Updates created panel object**

```
...
    oObjSddProductPlate.Update
End Sub
```

Method `UpdateObject` updates the created panel.

Fig.1: Plate




# Creating a SDD Member

This use case primarily focuses on the methodology to create a SDD member.

---

Related Topics

Structure Design Object Model Map

Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcCreateMember.3dxml`, `CAAScdSddUcSteel_A42.3dxml`, `CAAScdSddUcWT18x179_5.3dxml`, `CAAScdSddUcCGR.3dxml` and `CAAScdSddUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcCreateMemberSource.htm

This use case can be divided in 11 steps:

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Part object
3. Retrieves Service Manger (RfgService)
4. Retrieves SddFactory for creating SDD objects
5. Creates a member using two surfaces
6. Creates a member using a point and a limit
7. Creates a member using a point and length
8. Creates a member using a curve
9. Creates a member using a surface and two members(crvs)
10. Creates a member using two points
11. Updates the Part object

1. **Searches and opens model which is named as "SddProduct"**

As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim SDDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SDDPrdEditor
```

```
...
```

The function **OpenProduct** returns **SDDPrdEditor**, a Editor object. After searching and opening of SDD model from underlying database the current active editor

2. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Dim product1Service  As  PLMProductService
  Set product1Service = DDPrdEditor.GetService("PLMProductService")
  Dim ObjVPMRootOccurrence  As  VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference  As  VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances  As  VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

3. **Retrieves Service manager (RfgService)**

   In this step UC retrieves `RfgService`.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

`GetService` method returns `RfgService`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

4. **Retrieve SddFactory for creating SDD objects**

   In this step Uc retrieves the SddFactory for creating SDD objects.

```
...
  Dim ObjSddFactory  As  SddFactory
  SDDProdSel.Add ObjVPMRootOccurrence
  Set ObjSddFactory = SDDProdSel.FindObject("CATIASddFactory")
...
```

In this step Uc retrieves the SddFactory by calling`FindObject` method as shown above.

5. **Creates a member using two surfaces**

   Call `CreateMemberSurfSurf` method to create member using two surfaces. `CreateMemberSurfSurf` method takes SddFactory as input parameter and it returns cr in `ObjSddProdMemberSurfSurf`.

```
...
        Dim ObjSddProdMemberSurfSurf As SddProductMember
        CreateMemberSurfSurf ObjSddFactory, ObjSddProdMemberSurfSurf
...
```

The method `CreateMemberSurfSurf` is detailed as in the below sub steps.

   i. **Creates a empty member and sets category, material, profile type and section parameters**

```
Sub CreateMemberSurfSurf(iObjSddFactory As SddFactory, oObjProdMember As SddProductMember)
  CreateMemberAndSetData iObjSddFactory, "SddMember", catStrProfileModeSurfSurf, "Steel A42", "WT18x179.5", "catStrTopCenter"
...
```

   In this step Uc calls a method `CreateMemberAndSetData` which creates a empty member and sets some of the properties of the member and return the creat method sets category, profile type, material, section name and anchor point of the member. Here iObjSddFactory is the SddFactory, catStrProfileModeSurfS (catStrProfileModeSurfSurf means profile is created on intersection of two surfaces), "Steel A42" is material name, "WT18x179.5" is section name and "cat

   ii. **Retrieves StrProfileLimitMngt and sets profile limits**

```
...
    Dim ObjStrProfileLimitMngt As StrProfileLimitMngt
    Set ObjStrProfileLimitMngt = ObjSddMember.StrProfileLimitMngt
    'set Start limit
    Set ObjStartLimit = Manager.GetReferencePlane(ObjPart, 1, "DECK.8")
    Set StartLimit = ObjPart.CreateReferenceFromObject(ObjStartLimit)
    ObjStrProfileLimitMngt.SetLimitingObject 1, StartLimit
    'set End limit
    Set ObjEndLimit = Manager.GetReferencePlane(ObjPart, 1, "DECK.3")
    Set EndLimit = ObjPart.CreateReferenceFromObject(ObjEndLimit)
    ObjStrProfileLimitMngt.SetLimitingObject 2, EndLimit
...
```

   In this step Uc retrieves object of type `StrProfileLimitMngt` in `ObjStrProfileLimitMngt`. Reference to the plane DECK.8 is retrieved in `StartLimit` by Method `GetReferencePlane` takes 3 input parameters. First parameter is the part from which plane is, second is the index of the plane system (DECK = 1, ' name of the plane. To set start limit, `SetLimitingObject` method is called. It takes two inputs, **1** and **StartLimit**. here 1 means the limit is for Start of profi DECK.3 is set as the end limit for the profile. To set the End limit `SetLimitingObject` method takes first parameter as **2**.

   iii. **Retrieves StrProfileSurfSurf and sets surfaces for member creation**

```
...
    'Get StrProfileSurfSurf object
    Dim ObjStrProfileSurfSurf As StrProfileSurfSurf
    Set ObjStrProfileSurfSurf = ObjSddMember.StrProfileSurfSurf
    'Set FirstSurface
    Set ObjRefFirstSurf = Manager.GetReferencePlane(ObjPart, 2, "CROSS.8")
    Set RefFirstSurf = ObjPart.CreateReferenceFromObject(ObjRefFirstSurf)
    ObjStrProfileSurfSurf.FirstSurface = RefFirstSurf
    'Set SecondSurface
    Set ObjRefSecondSurf = Manager.GetReferencePlane(ObjPart, 3, "LONG.-4")
```

```
    Set RefSecondSurf = ObjPart.CreateReferenceFromObject(ObjRefSecondSurf)
    ObjStrProfileSurfSurf.SecondSurface = RefSecondSurf
...
```

This step sets two surfaces FirstSurface and SecondSurface. Member will be created at the intersection of these two surfaces. Uc first retrieves the object of FirstSurface, it retrieves the reference to the CROSS.8 plane in `RefFirstSurf` and assign it to the `FirstSurface` property of the `ObjStrProfileSurfSurf` retrieves the reference to the LONG.-4 plane in `RefSecondSurf` and assign it to the `SecondSurface` property of the `ObjStrProfileSurfSurf` object.

iv. **Updates created member object**

```
...
    oObjProdMember.Update
  End Sub
```

Method `Update` updates the created member.

6. **Creates a member using a point and a limit**

Call `CreateMemberPtLimit` method to create member using a point and a limit. `CreateMemberPtLimit` method takes SddFactory as input parameter and it return `ObjMemberPtLimit`.

```
...
  Dim ObjSddProdMemberPtLimit As SddProductMember
  CreateMemberPtLimit ObjSddFactory, ObjSddProdMemberPtLimit
...
```

The method `CreateMemberPtLimit` is detailed as in the below sub steps.

i. **Creates a empty member and sets category, material, profile type and section parameters**

```
Sub CreateMemberPtLimit(iObjSddFactory As SddFactory, oObjProdMember As SddProductMember)
  CreateMemberAndSetData iObjSddFactory, "SddMember", catStrProfileModePtLimit, "Steel A42", "WT18x179.5", "catStrTopCenter",
...
```

In this step Uc calls a method `CreateMemberAndSetData` which creates a empty member and sets some of the properties of the member and return the creat parameter of this method is catStrProfileModePtLimit. Here catStrProfileModePtLimit means profile is created using a point and a limit.

ii. **Retrieves a point and reference to it for member creation**

```
...
    Set PointObj = ObjPart.FindObjectByName("Point.1")
    Set RefPoint = ObjPart.CreateReferenceFromObject(PointObj)
...
```

In this step Uc retrieves a point and reference to this point is stored in `RefPoint`.

iii. **Retrieves StrProfilePtLimit and sets point and limit and direction**

```
...
    'Get StrProfilePtLimit object
    Dim ObjStrProfilePtLimit As StrProfilePtLimit
    Set ObjStrProfilePtLimit = ObjSddMember.StrProfilePtLimit
    'set point
    ObjStrProfilePtLimit.StartPoint = RefPoint
    'set limit
    Set ObjMemberLimitRef = Manager.GetReferencePlane(ObjPart, 1, "DECK.4")
    Set MemberLimitRef = ObjPart.CreateReferenceFromObject(ObjMemberLimitRef)
    ObjStrProfilePtLimit.UpToLimit = MemberLimitRef
    'set direction
    ObjStrProfilePtLimit.Direction = MemberLimitRef
...
```

In this step object of type `StrProfilePtLimit` is retrieved in `ObjStrProfilePtLimit`. Then `RefPoint`, which is created in the step above is assigned to th Uc needs to set the limit. To set the limit it retrieves the reference to plane DECK.4 in `MemberLimitRef` and assigns it to the `UpToLimit` property. Same pla also.

iv. **Updates created member object**

```
...
    oObjProdMember.Update
  End Sub
```

Method `Update` updates the created member.

7. **Creates a member using a point and length**

Call `CreateMemberPtLength` method to create member using a point and length. `CreateMemberPtLength` method takes a SddFactory object as input parameter a parameter in `ObjSddProdMemberPtLength`.

```
...
  Dim ObjSddProdMemberPtLength As SddProductMember
  CreateMemberPtLength ObjSddFactory, ObjSddProdMemberPtLength
...
```

The method `CreateMemberPtLength` is detailed as in the below sub steps.

i. **Creates a empty member and sets category, material, profile type and section parameters**

```
Sub CreateMemberPtLength(iObjSddFactory As SddFactory, oObjProdMember As SddProductMember)
  CreateMemberAndSetData iObjSddFactory, "SldMember", catStrProfileModePtLength, "Steel A42", "WT18x179.5", "catStrTopCenter"
...
```

In this step Uc calls a method `CreateMemberAndSetData` which creates a empty member and sets some of the properties of the member and return the creat parameter of this method is catStrProfileModePtLength. Here catStrProfileModePtLength means profile is created using a point and length.

ii. **Retrieves a point and reference to it for member creation**

```
...
    Set PointObj = ObjPart.FindObjectByName("Point.2")
    Set RefPoint = ObjPart.CreateReferenceFromObject(PointObj)
```

In this step Uc retrieves a point using method `FindObjectByName` and reference to this point is stored in `RefPoint`.

iii. **Retrieves StrProfilePtLength and sets point and direction and length**

```
...
    'Get StrProfilePtLength object
    Dim ObjStrProfilePtLength As StrProfilePtLength
    Set ObjStrProfilePtLength = ObjSddMember.StrProfilePtLength
    'set point
    ObjStrProfilePtLength.StartPoint = RefPoint
    'set direction
    Set RefMemberDirection = Manager.GetReferencePlane(ObjPart, 1, "DECK.7")
    ObjStrProfilePtLength.Direction = RefMemberDirection
    'set length
    Set LengthParm = ObjStrProfilePtLength.GetLength
    LengthParm.ValuateFromString ("20000mm")
...
```

In this step object of type `StrProfilePtLength` is retrieved in `ObjStrProfilePtLength`. Then `RefPoint`, which is created in the step above is assigned to reference to the plane DECK.7 and is assigned to the `Direction` property. To set the length it retrieves the `LengthParm` by using method `GetLength` and its `ValuateFromString` method is used.

iv. **Updates created member object**

```
...
    oObjProdMember.Update
End Sub
```

Method `Update` updates the created member.

8. **Creates a member using a curve**

Call `CreateMemberCrv` method to create member using a curve. `CreateMemberCrv` method takes a SddFactory object as input parameter and it returns created me `ObjSddProdMemberCrv`.

```
...
  Dim ObjSddProdMemberCrv As SddProductMember
  CreateMemberCrv ObjSddFactory, ObjSddProdMemberCrv
...
```

The method `CreateMemberCrv` is detailed as in the below sub steps.

i. **Creates a empty member and sets category, material, profile type and section parameters**

```
Sub CreateMemberCrv(iObjSddFactory As SddFactory, oObjProdMember As SddProductMember)
  CreateMemberAndSetData iObjSddFactory, "SddMember", catStrProfileModeCrv, "Steel A42", "WT18x179.5", "catStrTopCenter", oOb
...
```

In this step Uc calls a method `CreateMemberAndSetData` which creates a empty member and sets some of the properties of the member and return the creat parameter of this method is catStrProfileModeCrv. Here catStrProfileModeCrv means profile is created using a curve.

ii. **Retrieves a line and reference to it for member creation**

```
...
    Set ObjHybridShapeLinePtPt = ObjPart.FindObjectByName("Line.1")
    Set RefLine = ObjPart.CreateReferenceFromObject(ObjHybridShapeLinePtPt)
...
```

In this step Uc retrieves a line using method `FindObjectByName` and reference to this line is stored in `RefLine`.

iii. **Retrieves StrProfileCrv and sets Curve**

```
...
    'Get StrProfileCrv object
    Dim ObjStrProfileCrv As StrProfileCrv
    Set ObjStrProfileCrv = ObjSddMember.StrProfileCrv
    'set the curve
    ObjStrProfileCrv.Curve = RefLine
...
```

In this step object of type `StrProfileCrv` is retrieved in `ObjStrProfileCrv`. Then `RefLine`, which is retrieved in the step above is assigned to the `Curve` p member is created.

iv. **Updates created member object**

```
...
    oObjProdMember.Update
End Sub
```

Method `Update` updates the created member.

9. **Creates a member using a surface and two members(curves)**

Call `CreateMemberSurf2Crvs` method to create member using a surface and two curves. `CreateMemberSurf2Crvs` method takes a SddFactory object, and two m created member as output parameter in `ObjMemberSurf2Crv`.

```
...
  Dim ObjSddProdMemberSurf2Crv As SddProductMember
```

```
  CreateMemberSurf2Crvs ObjSddFactory, ObjSddProdMemberPtLimit, ObjSddProdMemberPtLength, ObjSddProdMemberSurf2Crv
...
```

The method `CreateMemberSurf2Crvs` is detailed as in the below sub steps.

i. **Creates a empty member and sets category, material, profile type and section parameters**

```
Sub CreateMemberSurf2Crvs(iObjSddFactory As SddFactory, iFirstMember As SddProductMember, iSecondMember As SddProductMember,
  CreateMemberAndSetData iObjSddFactory, "SddMember", catStrProfileModeSurf2Crvs, "Steel A42", "WT18x179.5", "catStrTopCenter
...
```

In this step Uc calls a method `CreateMemberAndSetData` which creates a empty member under given GeometricalSet and sets some of the properties of the oObjProdMember. Third parameter of this method is catStrProfileModeSurf2Crvs. Here catStrProfileModeSurf2Crvs means profile is created using a surfac

ii. **Retrieves StrProfileSurf2Crvs and sets a surface and two curves**

```
...
    'Get StrProfileSurf2Crvs object
    Dim ObjStrProfileSurf2Crvs As StrProfileSurf2Crvs
    Set ObjStrProfileSurf2Crvs = ObjSddMember.StrProfileSurf2Crvs
    'Set Surface
    Set ObjRefSurface = Manager.GetReferencePlane(ObjPart, 1, "DECK.3")
    Set RefSurface = ObjPart.CreateReferenceFromObject(ObjRefSurface)
    ObjStrProfileSurf2Crvs.Surface = RefSurface
    'Set FirstCurve
    Set RefFirstCrv = iFirstMember.GetDelimitedSupport
    ObjStrProfileSurf2Crvs.FirstCurve = RefFirstCrv
    'Set SecondCurve
    Set RefSecondCrv = iSecondMember.GetDelimitedSupport
    ObjStrProfileSurf2Crvs.SecondCurve = RefSecondCrv
...
```

This step sets a surface and two curves. Member will be created on the surface and between two given curves. Uc first retrieves the object of type `StrProfi` `ObjStrProfileSurf2Crvs`. To set the surface, it first retrieves the reference to the plane DECK.3 and assigns it to the `Surface` property. To set the FirstCu `iFirstMember` by using method `GetDelimitedSupport` and assigns it to the `FirstCurve` property. In the same manner, it retrieves the delimited support o `SecondCurve` property.

iii. **Updates created member object**

```
...
    oObjProdMember.Update
End Sub
```

Method `Update` updates the created member.

10. **Creates a member using two points**

Call `CreateMemberPtPt` method to create member using two points. `CreateMemberPtPt` method takes a SddFactory object as input parameter and it returns creat `ObjMemberPtPt`.

```
...
  Dim ObjSddProdMemberPtPt As SddProductMember
  CreateMemberPtPt ObjSddFactory, ObjSddProdMemberPtPt
...
```

The method `CreateMemberPtPt` is detailed as in the below sub steps.

i. **Creates a empty member and sets category, material, profile type and section parameters**

```
Sub CreateMemberPtPt(iObjSddFactory As SddFactory, oObjProdMember As SddProductMember)
  CreateMemberAndSetData iObjSddFactory, "SddMember", catStrProfileModePts, "Steel A42", "WT18x179.5", "catStrTopCenter", oOb
...
```

In this step Uc calls a method `CreateMemberAndSetData` which creates a empty member and sets some of the properties of the member and return the creat parameter of this method is catStrProfileModePts. Here catStrProfileModePts means profile is created using two points.

ii. **Retrieves two points and references to these points for member creation.**

```
...
    'Create Start point
    Set StartPoint = ObjPart.FindObjectByName("Point.3")
    Set RefStartPoint = ObjPart.CreateReferenceFromObject(StartPoint)
    'Create End point
    Set EndPoint = FindObjectByName("Point.4")
    Set RefEndPoint = ObjPart.CreateReferenceFromObject(EndPoint)
...
```

This retrieves two points and stores its references in `RefStartPoint` and `RefEndPoint`. Member will be created between these two points.

iii. **Retrieves StrProfilePtPt object and sets StartPoint and EndPoint**

```
...
    'Get StrProfilePtPt object
    Dim ObjStrProfilePtPt As StrProfilePtPt
    Set ObjStrProfilePtPt = ObjSddMember.StrProfilePtPt
    'Set StartPoint
    ObjStrProfilePtPt.StartPoint = RefStartPoint
    'Set EndPoint
    ObjStrProfilePtPt.EndPoint = RefEndPoint
...
```

This step sets two points. Member will be created in between these two points. Uc first retrieves the object of type `StrProfilePtPt` in `ObjStrProfilePtPt` `RefStartPoint`(reference to the point) to the `StartPoint` property. To set the EndPoint, it assigns the `RefEndPoint`(refernce to the point) to the `EndPoint`

iv. **Updates created member object**

```
...
```

```
    oObjProdMember.Update
End Sub
```

Method `Update` updates the created member.

11. **Updates the Part object**

```
...
  ObjPart.Update
End Sub
```

`Update` method updates the `ObjPart`

**Detailed steps of method CreateMemberAndSetData**

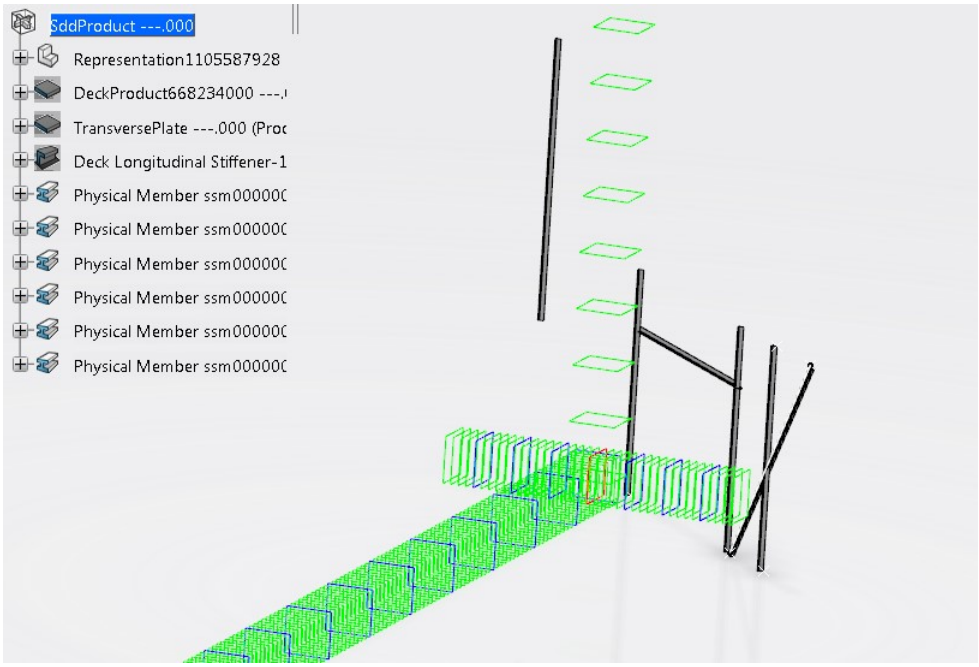
- **CreateMemberAndSetData method**

This method creates a new empty member, sets some properties of the member and returns the created member in `oObjMember` object. Properties which will set by material, secion name and section's anchor point.

```
Sub CreateMemberAndSetData(iObjSddFactory As SddFactory, iCategory As String, iProfileType As CATStrProfileMode, iMaterial As Stri
                           iSectionName As String, iAnchorPoint As String, oObjProdMember As SddProductMember)

'   1- creates a empty member
    Set oObjProdMember = iObjSddFactory.AddProductMember
    Dim ObjSddMember As SddMember
    Set ObjSddMember = oObjProdMember.SddMember

'   2- Retrieves StrCategoryMngt object and sets category
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = ObjSddMember.StrCategoryMngt
    ObjStrCategoryMngt.SetCategory iCategory

'   3- Sets profile type
    ObjSddMember.ProfileType = iProfileType

'   4- Retrieves StrMaterialMngt object and sets material
    Dim ObjMaterialMngt As StrMaterialMngt
    Set ObjMaterialMngt = oObjProdMember.StrMaterialMngt
    ObjMaterialMngt.SetMaterial iMaterial

'   5- Retrieves StrSectionMngt object and sets section parameters
    Dim ObjStrSectionMngt As StrSectionMngt
    Set ObjStrSectionMngt = ObjSddMember.StrSectionMngt
    ObjStrSectionMngt.SetSectionName iSectionName
    ObjStrSectionMngt.AnchorPoint = iAnchorPoint

End Sub
```

In the first step of this method it calls method `AddProductMember` to create a new empty member and stores it in `oObjProductMember` which is a output paramete In the second step, it retrieves the object of type `StrCategoryMngt` in `ObjStrCategoryMngt` and calls `SetCategory` method to set the category. In the third step, `iProfileType`. In the fourth step, it retrieves the object of type `StrMaterialMngt` in `ObjStrMaterialMngt` and calls `SetMaterial` method to set the material. In `StrSectionMngt` in `ObjStrSectionMngt` and calls `SetSectionName` method to set the secion name and sets the property `AnchorPoint` with `iAnchorPoint`.

Fig.1: Member




# Creating an SDD Stiffener

This use case primarily focuses on the methodology to create a SDD stiffener.

Related Topics

Structure Design Object

Before you begin: Note that:

Model Map
Launching an Automation Use Case

- You should first launch CATIA and import the `CAAScdSddUcCreateStiffener.3dxml`, `CAAScdSddUcSteel_A42.3dxml`, `CAAScdSddUcWT18x179_5.3dxml`, `CAAScdSddUcCGR.3dxml` and `CAAScdSddUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcCreateStiffenerSource.htm

This use case can be divided in nine steps:

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves Service manager
5. Retrieves a SddPlate
6. Retrieves SddStiffenerMngt
7. Creates a Stiffener
8. Edits created Stiffener
9. Update the Part object

1. **Searches and opens model which is named as "SddProduct"**

   As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim SDDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SDDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SDDPrdEditor`**, a Editor object. After searching and opening of SDD model from underlying database the current active editor in **`SDDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves the representation variable. To retrieve the Selection object **`SDDPrdEditor`** is used.

```
 ...
  Set SDDProdSel = SDDPrdEditor.Selection
 ...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Dim product1Service As PLMProductService
  Set product1Service = SDDPrdEditor.GetService("PLMProductService")
  Dim ObjVPMRootOccurrence As VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference As VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances As VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

4. **Retrieves Service Manager**

   In this step UC retrieves object of RfgService.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

   *GetService* method returns the service `RfgService` in `Manager` variable. This service provides various services.

5. **Retrieves a SddPlate object**

   In this step UC retrieves a SddPlate from SddProductPlate which is retrieved using Selection object.

```
...
  Set ListOfInstances = ObjVPMReference.Instances
  Set PlateRef = ListOfInstances.Item(1).ReferenceInstanceOf
  Set PlateRepInstances = PlateRef.RepInstances
  Set PlateRepInstReference = PlateRepInstances.Item(1).ReferenceInstanceOf
  Set PlatePart = PlateRepInstReference.GetItem("Part")
  Dim ObjSddProductPlate As SddProductPlate
  SFDProdSel.Add PlateRef
  Set ObjSddProductPlate = SDDProdSel.FindObject("CATIASddProductPlate")
  Dim ObjSddPlate As SddPlate
  Set ObjSddPlate = ObjSddProductPlate.SddPlate
...
```

   In above lines, *FindObject* method finds the object "CATIASddProductPlate" and returns to it. To retrieve `SddPlate` object call *SddPlate* property as shown above give the `SddPlate` object.

6. **Retrieves a SddStiffenerMngt object**

   In this step UC retrieves a SddStiffenerMngt object using Selection object.

```
...
  SFDProdSel.Add ObjVPMRootOccurrence
  Dim ObjSddStiffenerMngt As SddStiffenerMngt
  Set ObjSddStiffenerMngt = SFDProdSel.FindObject("CATIASddStiffenerMngt")
```

```
...
```

In above lines, *FindObject* method finds the object "CATIASddStiffenerMngt" and returns to it. This will give the `SddStiffenerMngt` object.

7. **Creates a Stiffener**

Now plate is available to create stiffener on it. Call `CreateStiffener` method to create stiffener. `CreateStiffener` method takes a panel and SddStiffenerMngt o input and created stiffener is returned in output parameter.

```
...
  Dim ObjSddProductStiffener As SddProductStiffener
  CreateStiffener ObjSddPlate, ObjSddStiffenerMngt, ObjSddProductStiffener
...
```

The method `CreateStiffener` is detailed as in the below sub steps.

i. Create a new empty stiffener by the method `AddStiffener` from object `SddStiffenerMngt`

```
Sub CreateStiffener(iObjSddPlate As SddPlate, iObjSddStiffenerMngt As SddStiffenerMngt, oObjSddProductStiffener As SddProduct
  'Add Stiffener
  Set oObjSddProductStiffener = iObjSddStiffenerMngt.AddStiffener
...
```

On `ObjSddStiffenerMngt` object `AddStiffener` method is called to create the empty stiffener. Now Uc needs to set the different properties of the stiffener material, category etc.

ii. Set different properties of the stiffener like material. Retrieve the `StrMaterialMngt` object from `SddProductStiffener` object and set the material for the s using `SetMaterial` method.

```
...
  'Get StrMaterialMngt object
    Dim ObjMaterialMngt As StrMaterialMngt
    Set ObjMaterialMngt = oObjSddProductStiffener.StrMaterialMngt
  'Set material of the stiffener
    ObjMaterialMngt.SetMaterial("Steel A42")
...
```

`ObjMaterialMngt` object is of type `StrMaterialMngt`. It is retrieved from the stiffener `oObjSddProductStiffener`. `SetMaterial` method is called to set t on object `ObjMaterialMngt`.

iii. To set the category retrieve the `StrCategoryMngt` object from `SddStiffener` object and set category using `SetCategory` method.

```
...
  'Get StrCategoryMngt object
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = ObjSddStiffener.StrCategoryMngt
  'Set category of the stiffener
    ObjStrCategoryMngt.SetCategory("SddStiffener")
...
```

iv. Sets the `ProfileType` property of the stiffener to catStrProfileModeSurfSurf (here catStrProfileModeSurfSurf means profile is created with the intersection surfaces).

```
...
  'Set type of the stiffener
    ObjSddStiffener.ProfileType = catStrProfileModeSurfSurf
...
```

v. Retrieves the `StrSectionMngt` object from the created stiffener object and sets the different section parameters like section name, anchor point, web orienta orientation.

```
...
  'Get StrSectionMngt object
    Dim ObjStrSectionMngt As StrSectionMngt
    Set ObjStrSectionMngt = ObjSddStiffener.StrSectionMngt
  'Set different section parameters
    ObjStrSectionMngt.SetSectionName ("WT18x179.5")
    ObjStrSectionMngt.AnchorPoint = "catStrTopCenter"
    ObjStrSectionMngt.WebOrientation = 1
    ObjStrSectionMngt.FlangeOrientation = 1

    ObjStrSectionMngt.WebOrientation = 1
    ObjStrSectionMngt.AngleMode = 0
...
```

`ObjStrSectionMngt` object which is of type `StrSectionMngt` is used to set the different section properties of the stiffener. `SetSectionName` method sets th name for the stiffener. `AnchorPoint` property is used to set anchor point to "catStrTopCenter". Similarly `WebOrientation` and `FlangeOrientation` prope to 1.

vi. Retrieves the `StrProfileLimitMngt` object from the created stiffener object and sets the Start limit and End limit of the stiffener.

```
...
  'Get StrProfileLimitMngt object
    Dim ObjStrProfileLimitMngt As StrProfileLimitMngt
    Set ObjStrProfileLimitMngt = ObjSddStiffener.StrProfileLimitMngt
  'Set the profile limits
    Set ObjStartLimit = Manager.GetReferencePlane(ObjPart, 2, "CROSS.70")
    Set StartLimit = ObjPart.CreateReferenceFromObject(ObjStartLimit)
    ObjStrProfileLimitMngt.SetLimitingObject 1, StartLimit
    Set ObjEndLimit = Manager.GetReferencePlane(ObjPart, 2, "CROSS.110")
    Set EndLimit = ObjPart.CreateReferenceFromObject(ObjEndLimit)
    ObjStrProfileLimitMngt.SetLimitingObject 2, EndLimit
...
```

`ObjStrProfileLimitMngt` is retrieved from the `ObjSddStiffener` object. Variable `StartLimit` and `EndLimit` are of type `Reference` which contains refe planes "CROSS.70" and "CROSS.110" respectively. Plane is retrieved using `GetReferencePlane` method on `Manager` object. `Manager` object is of type `Rf` Reference to this plane retrieved is created using `CreateReferenceFromObject` method. `ObjStrProfileLimitMngt` calls `SetLimitingObject` method to

and end limit of the of the stiffener. `SetLimitingObject` method has two input parameters. First input defines whether it is start of the stiffener or end of th Second input is the limiting object.

vii. Retrieves the `StrProfileSurfSurf` object from the created stiffener object and sets two surfaces which are intersecting with each other. At the intersection two surfaces stiffener will be created.

```
...
  'Get StrProfileSurfSurf object
    Dim ObjStrProfileSurfSurf As StrProfileSurfSurf
    Set ObjStrProfileSurfSurf = ObjSddStiffener.StrProfileSurfSurf
  'Set two surfaces
    Dim PlateReference As Reference
    Set PlateReference = PlatePart.CreateReferenceFromObject(ObjSddPlate)
    ObjStrProfileSurfSurf.FirstSurface = PlateReference
    Set ObjWebSupport = Manager.GetReferencePlane(ObjPart, 3, "LONG.-4")
    Set WebSupport = ObjPart.CreateReferenceFromObject(ObjWebSupport)
    ObjStrProfileSurfSurf.SecondSurface = WebSupport
...
```

`ObjStrProfileSurfSurf` object is retrieved from `ObjSddStiffener` object. Here Uc is setting two intersecting surfaces for stiffener creation. To set first s retrieves the reference of SddPlate object in `PlateReference` using method `CreateReferenceFromObject`. Then `PlateReference` is assigned to the `Firs` property of the `ObjStrProfileSurfSurf` object. Similarly `SecondSurface` is set.

viii. Updates the created stiffener.

```
...
  oObjSddProductStiffener.Update
End Sub
...
```

8. **Edits created stiffener**

In this step UC edits stiffener's some of the data like it changes the support of the stiffener then it changes anchor point, web orientation and flange orientation. Fo stiffener data `EditStiffener` method is called.

```
...
  'Edits the stiffener
  EditStiffener ObjSddProductStiffener
...
```

The method `EditStiffener` is detailed as in the below sub steps.

i. Retrieves the `SddStiffener` object from the SddProductStiffener object.

```
Sub EditStiffener(iObjSddProductStiffener As SddProductStiffener)
  'Get SddStiffener object
    Dim ObjSddStiffener As SddStiffener
    Set ObjSddStiffener = iObjSddProductStiffener.SddStiffener
...
```

ii. Retrieves the `StrProfileSurfSurf` object from the stiffener object. Then changes the support of the Stiffener to LONG.0.

```
...
    Dim ObjSddStiffener As SddStiffener
    Set ObjSddStiffener = iObjSddProductStiffener.SddStiffener
  'Get StrProfileSurfSurf object
    Dim ObjStrProfileSurfSurf As StrProfileSurfSurf
    Set ObjStrProfileSurfSurf = ObjSddStiffener.StrProfileSurfSurf
  'Change the web support of the stiffener
    Set ObjWebSupport = Manager.GetReferencePlane(ObjPart, 3, "LONG.0")
    Set WebSupport = ObjPart.CreateReferenceFromObject(ObjWebSupport)
    ObjStrProfileSurfSurf.SecondSurface = WebSupport
...
```

iii. Retrieves the `StrSectionMngt` object from the created stiffener object and changes anchor point, web orientation and flange orientation.

```
...
  'Get StrSectionMngt object
    Dim ObjStrSectionMngt As StrSectionMngt
    Set ObjStrSectionMngt = ObjSddStiffener.StrSectionMngt
  'Change AnchorPoint, WebOrientation, FlangeOrientation
    ObjStrSectionMngt.AnchorPoint = "catStrTopLeft"
    ObjStrSectionMngt.WebOrientation = -1
    ObjStrSectionMngt.FlangeOrientation = -1
...
```

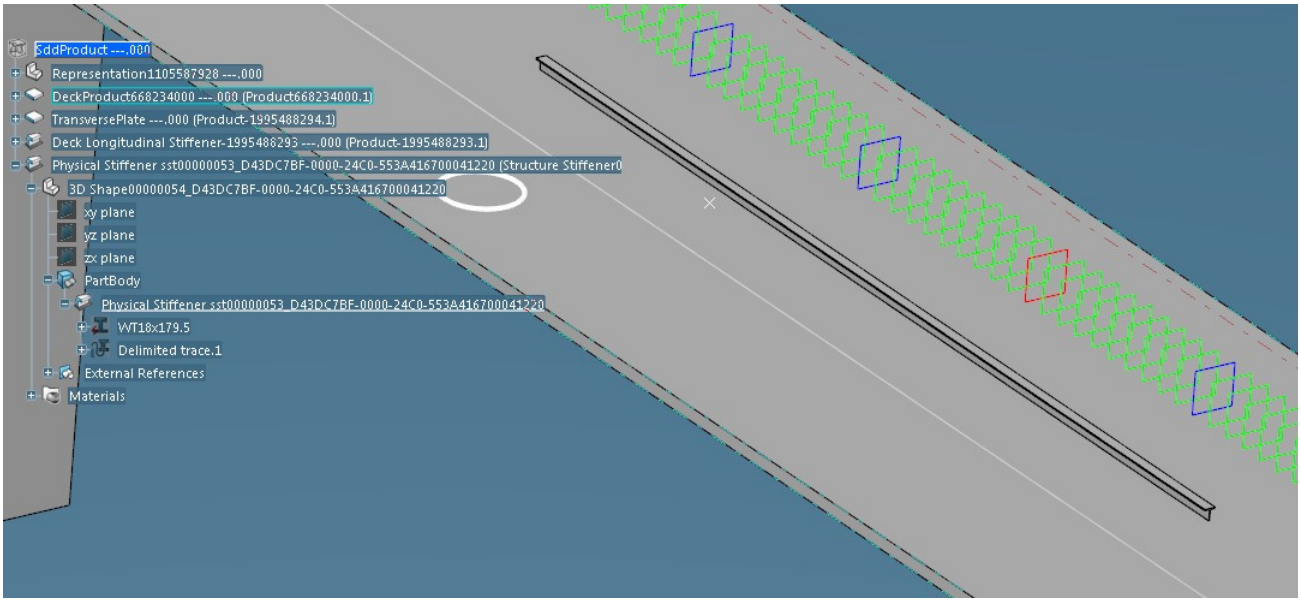
iv. Updates the stiffener object.

```
...
  'Update the stiffener
    iObjSddProductStiffener.Update
End Sub
...
```

9. **Update the Part object**

```
...
   ObjPart.Update
End Sub
```

Fig.1: Stiffener with Limits




# Creating an SDD Opening using 3D Objects

This use case primarily focuses on the methodology to create a SDD Opening using 3D object.

Related Topics
Structure Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcCreateOpening3DObject.3dxml`, `CAAScdSddUcCGR.3dxml` and `CAAScdSddUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcCreateOpening3DObjectSource.htm

This use case can be divided in seven steps:

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves Service manager
5. Retrieves a Plate object
6. Creates opening
7. Removes opening
8. Updates the Part object

1. **Searches and opens model which is named as "SddProduct"**

   As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim SDDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SDDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SDDPrdEditor`**, a Editor object. After searching and opening of SDD model from underlying database the current active editor is returned in **`SDDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **`SDDPrdEditor`** is used.

```
 ...
    Set SDDProdSel = SDDPrdEditor.Selection
 ...
```

3. **Retrieves the part object**

   In this step UC retrieves part object.

```
...

    Dim product1Service As PLMProductService
    Set product1Service = SDDPrdEditor.GetService("PLMProductService")
    Dim ObjVPMRootOccurrence As VPMRootOccurrence
    Set ObjVPMRootOccurrence = product1Service.RootOccurrence
    Dim ObjVPMReference As VPMReference
    Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
    Dim ObjVPMRepInstances As VPMRepInstances
    Set ObjVPMRepInstances = ObjVPMReference.RepInstances
    Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
    Set ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

4. **Retrieves Service manager**

   In this step UC retrieves Service manager variable.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

5. **Retrieves a Plate object**

   In this step UC retrieves plate object.

```
...

    Set ListOfInstances = ObjVPMReference.Instances
    Set PlateRef = ListOfInstances.Item(1).ReferenceInstanceOf
    Dim ObjSddProductPlate As SddProductPlate
    SDDProdSel.Add PlateRef
    Set ObjSddProductPlate = SDDProdSel.FindObject("CATIASddProductPlate")
    Dim ObjSddPlate As SddPlate
    Set ObjSddPlate = ObjSddProductPlate.SddPlate
...
```

   `FindObject` method finds object whose type is "SddProductPlate" and returns it. To retrieve `SddPlate` object from `ObjSddProductPlate` call `SddPlate` property as shown above. This will return the `SddPlate` object variable.

6. **Creates opening**

   Now, plate is available to create opening on it. Call `CreateOpening3DObject` method to create opening on panel. `CreateOpening3DObject` method takes a SddPlate as input parameter and it returns created opening as output parameter in `ObjStrOpening`.

```
...
  Dim ObjStrOpening As StrOpening
  CreateOpening3DObject ObjSddPlate, ObjStrOpening
...
```

   The method `CreateOpening3DObject` is detailed as in the below sub steps.

   i. **Creates a cylinder, later which will be used for opening creation as a intersecting profile**

```
Sub CreateOpening3DObject(iObjSddPlate As SddPlate, oObjStrOpening As StrOpening)
    Set ObjHybridShapeCylinder = ObjPart.FindObjectByName("Cylinder.1")
    Set CylinderRef = ObjPart.CreateReferenceFromObject(ObjHybridShapeCylinder)

...
```

      In this step Uc creates a cylinder and stores its reference in `CylinderRef`. Later this created cylinder will be used for opening creation as a intersecting element.

   ii. **Retrieves `StrOpenings` and creates an opening with no properties set**

```
...
    'Get StrOpenings object
    Dim ObjStrOpenings As StrOpenings
    Set ObjStrOpenings = iObjSddPlate.GetOpenings(0)
    'Add opening
    Set oObjStrOpening = ObjStrOpenings.Add
...
```

      Object of `StrOpenings` is retrieved in `ObjStrOpenings`. Then method `Add` from `ObjStrOpenings`is called to create an opening. This creates an opening with not properties set. Now Uc sets different properties on opening to complete the creation of the opening.

   iii. **Sets opening type to catStrOpeningMode3DObject**

```
...
    oObjStrOpening.OpeningType = catStrOpeningMode3DObject
...
```

      `OpeningType` property is set to catStrOpeningMode3DObject. catStrOpeningMode3DObject defines that opening is created using a 3D object.

   iv. **Retrieves `StrCategoryMngt` and sets category**

```
...
    'Get StrCategoryMngt object
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = oObjStrOpening.StrCategoryMngt
    'set category
    ObjStrCategoryMngt.SetCategory "SddOpening"
...
```

      Object of `StrCategoryMngt` is retrieved in `ObjStrCategoryMngt`. Then method `SetCategory` is called to set the category.

   v. **Retrieves `StrOpening3DObject` and sets intersecting profile**

```
...
    'Get StrOpening3DObject object
    Dim ObjStrOpening3DObject As StrOpening3DObject
    Set ObjStrOpening3DObject = oObjStrOpening.StrOpening3DObject
    'Set intersecting element
    ObjStrOpening3DObject.IntersectingElement = CylinderRef
...
```

      Object of `StrOpening3DObject` is retrieved in `ObjStrOpening3DObject`. Then created `CylinderRef` is assigned to the property `IntersectingElement`. Opening will be created at the intersection of plate and cylinder (3D Object).

   vi. **Retrieves StrOpeningExtrusionMngt and sets forming extrusion mode**

```
...
    Set ObjStrOpeningExtrusionMngt = oObjStrOpening.StrOpeningExtrusionMngt
    ObjStrOpeningExtrusionMngt.ExtrusionMode = 1
...
```

In this step object forming mode is set. To set the forming mode, forming mode value is assigned to property `ExtrusionMode`. Here 1 defines Before Forming Extrusion Mode..

7. **Removes opening**

```
...
    Dim ObjStrOpenings As StrOpenings
    Set ObjStrOpenings = ObjSddPlate.GetOpenings(0)
    ObjStrOpenings.Remove ObjStrOpening
...
```

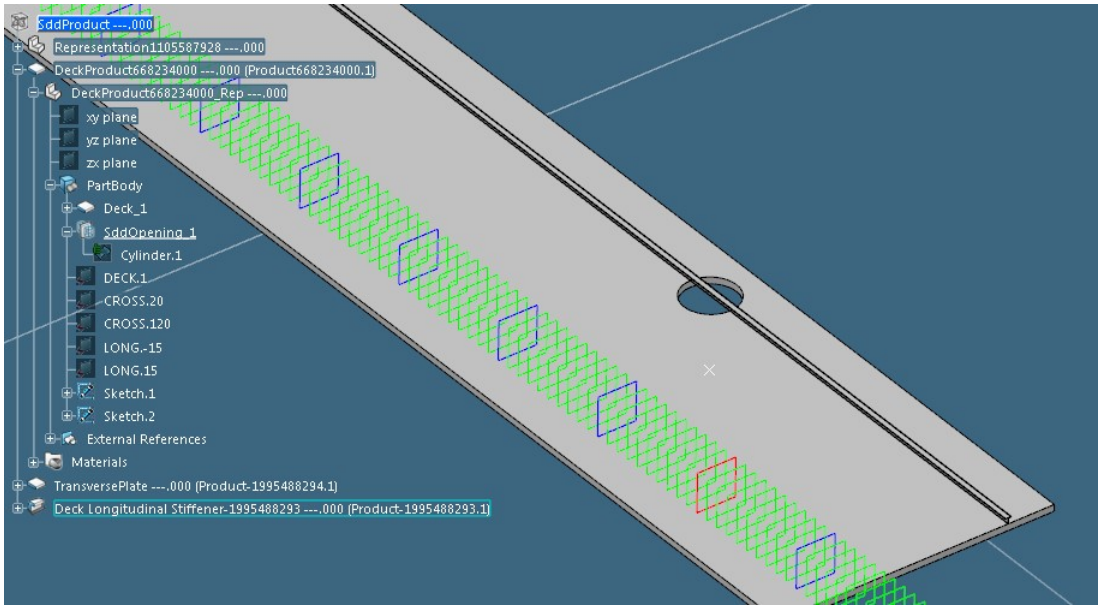
Object of `StrOpenings` is retrieved in `ObjStrOpenings`. Then on `ObjStrOpenings` object `Remove` method is called to remove the opening. This method takes a input parameter opening object to remove.

8. **Updates the Part object**

```
...
  ObjPart.Update
End Sub
```

`Update` method updates the `ObjPart`

Fig.1: Opening using 3D Object( before remove opening)




# Creating an SDD Opening Using a Sketch Profile

This use case primarily focuses on the methodology to create a SDD Opening using sketch profile.

---

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcCreateOpeningSketch.3dxml`, `CAAScdSddUcCGR.3dxml` and `CAAScdSddUcSR.3dxml` files supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Related Topics
*<topic1>*
*<topic2>*

Where to find the macro: CAAScdSddUcCreateOpeningSketchSource.htm

This use case can be divided in nine steps:

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves Services Manager
5. Retrieves Product Plate
6. Retrieves Sdd Plate
7. Retrieves profile for creating opening
8. Create Opening
9. Updates the Part object

1. **Searches and opens model which is named as "SddProduct"**

As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim SDDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SDDPrdEditor
...
```

The function **OpenProduct** returns **SDDPrdEditor**, a Editor object. After searching and opening of SDD model from underlying database the current active editor is returned in **SDDPrdEditor**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SDDProdSel variable. To retrieve the Selection object **SDDPrdEditor** is used.

```
...
   Set SFDProdSel = SDDPrdEditor.Selection
...
```

3. **Retrieves Part object**

   In this step UC retrieves part object.

```
...
    Dim product1Service As PLMProductService
    Set product1Service = SDDPrdEditor.GetService("PLMProductService")
    Dim ObjVPMRootOccurrence As VPMRootOccurrence
    Set ObjVPMRootOccurrence = product1Service.RootOccurrence
    Dim ObjVPMReference As VPMReference
    Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
    Dim ObjVPMRepInstances As VPMRepInstances
    Set ObjVPMRepInstances = ObjVPMReference.RepInstances
    Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
    Set ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

4. **Retrieves Service manager**

   In this step UC retrieves Service manager variable.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

   `GetService` method returns `RfgService`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

5. **Retrieves Product Plate**

   In this step UC retrieves a Product Plate.

```
...
  Set ListOfInstances = ObjVPMReference.Instances
  Set PlateRef = ListOfInstances.Item(1).ReferenceInstanceOf
  Set PlateRepInstances = PlateRef.RepInstances
  Set PlateRepInstReference = PlateRepInstances.Item(1).ReferenceInstanceOf
  Set PlatePart = PlateRepInstReference.GetItem("Part")
...
```

6. **Retrieves Sdd Plate**

   In this step UC retrieves a Sdd Plate.

```
...
  Dim ObjSddProductPlate  As  SddProductPlate
  SDDProdSel.Add PlateRef
  Set ObjSddProductPlate = SDDProdSel.FindObject("CATIASddProductPlate")
  Dim ObjSddPlate  As  SddPlate
  Set ObjSddPlate = ObjSddProductPlate.SddPlate
...
```

7. **Retrieves profile for creating opening**

```
...
  Set ProfileSketchOpeningUpToLast = PlatePart.FindObjectByName("Profile.1")
  Dim RefProfileSketchOpeningUpToLast  As  Reference
 Set RefProfileSketchOpeningUpToLast = PlatePart.CreateReferenceFromObject(ProfileSketchOpeningUpToLast)
...
```

   `FindObjectByName` method finds object whose name is "Profile.1" and returns reference to it. Here `RefProfileSketchOpeningUpToLast` is of type `Reference`. To retrieve the `RefProfileSketchOpeningUpToLast` object from `ProfileSketchOpeningUpToLast` object `CreateReferenceFromObject` method is used.

8. **Create Opening**

   Call `CreateOpeningSketchUpToLast` method to create opening on panel using sketch profile. `CreateOpeningSketchDimensions` method takes a SddPlate and a sketch profile as input parameters and it returns created opening as output parameter in `ObjStrOpeningWtLtMdUpToLast`.

```
...
  Dim ObjStrOpeningWtLtMdUpToLast As StrOpening
  CreateOpeningSketchUpToLast ObjSddPlate, RefProfileSketchOpeningUpToLast, ObjStrOpeningWtLtMdUpToLast
...
```

   The method `CreateOpeningSketchDimensions` is detailed as in the below sub steps.

   i. **Creates a opening with no properties set**

```
Sub CreateOpeningSketchUpToLast(iObjSddPlate As SddPlate, RefProfileSketch As Reference, oObjStrOpening As StrOpening)
  AddOpening iObjSddPlate, oObjStrOpening
...
```

   In this step Uc calls a method `AddOpening` which creates a opening with no properties set and returns it in output parameter `oObjStrOpening`. Uc sets different properties like catgory, type of opening, etc. of the opening in the subsequent steps.

   ii. **Sets opening type to catStrOpeningModeOutputProfile for sketch opening**

```
...
  SetOpeningType oObjStrOpening, catStrOpeningModeOutputProfile
...
```

In this step Uc calls a method `SetOpeningType`. This method takes a opening (oObjStrOpening) and opening type value (catStrOpeningModeOutputProfile) as input parameters. Here catStrOpeningModeOutputProfile is means the opening is sketch opening.

iii. **Sets category**

```
...
  SetCategory oObjStrOpening
...
```

In this step Uc calls a method `SetCategory`, it sets the category of the opening. This method takes the opening oObjStrOpening as input parameters.

iv. **Retrieves StrOpeningExtrusionMngt and sets forming extrusion mode**

```
...
    Set ObjStrOpeningExtrusionMngt = oObjStrOpening.StrOpeningExtrusionMngt
    ObjStrOpeningExtrusionMngt.ExtrusionMode = 1
...
```

In this step object forming mode is set. To set the forming mode, forming mode value is assigned to property `ExtrusionMode`. Here 1 defines Before Forming Extrusion Mode..

v. **Sets output profile**

```
...
  'set LimitMode to 1 for Dimensions
  SetOutPutProfile oObjStrOpening, RefProfileSketch, 0
...
```

In this step Uc calls a method `SetOutPutProfile`. It sets the output profile, direction and limit mode for the opening. This method takes a opening (oObjStrOpening), reference to sketch profile(RefProfileSketch) and limit mode value(0) as input parameters. Here 1 means limit mode is upto last.

9. **Updates the Part object**

```
...
  ObjPart.Update
End Sub
```

`Update` method updates the `ObjPart`

**Detailed steps of methods called in the use case**

- **AddOpening method**

```
Sub AddOpening(iObjSddPlate As SddPlate, oObjStrOpening As StrOpening)
    'Get StrOpenings object
    Dim ObjStrOpenings As StrOpenings
    Set ObjStrOpenings = iObjSddPlate.GetOpenings(0)
    'Add opening
    Set oObjStrOpening = ObjStrOpenings.Add
End Sub
```

Method `AddOpening`takes a plate object `iObjSddPlate` as input parameter and it returns created opening as output parameter in `oObjStrOpening`. In this method object of type `StrOpenings` is retrieved in `ObjStrOpenings`. Then on this object `Add` method is called to create an opening with no properties set.

- **SetOpeningType method**

```
Sub SetOpeningType(iObjStrOpening As StrOpening, iOpeningType As Long)
    'set opening type
    iObjStrOpening.OpeningType = iOpeningType
End Sub
```

Method `SetOpeningType` takes a opening object `iObjStrOpening` and opening type `iOpeningType` as input parameters. In this method, `iOpeningType` which is input to this method is assigned to `OpeningType` property.

- **SetCategory method**

```
Sub SetCategory(iObjStrOpening As StrOpening)
    'Get StrCategoryMngt object
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = iObjStrOpening.StrCategoryMngt
    'set catgory
    ObjStrCategoryMngt.SetCategory iCategory
End Sub
```

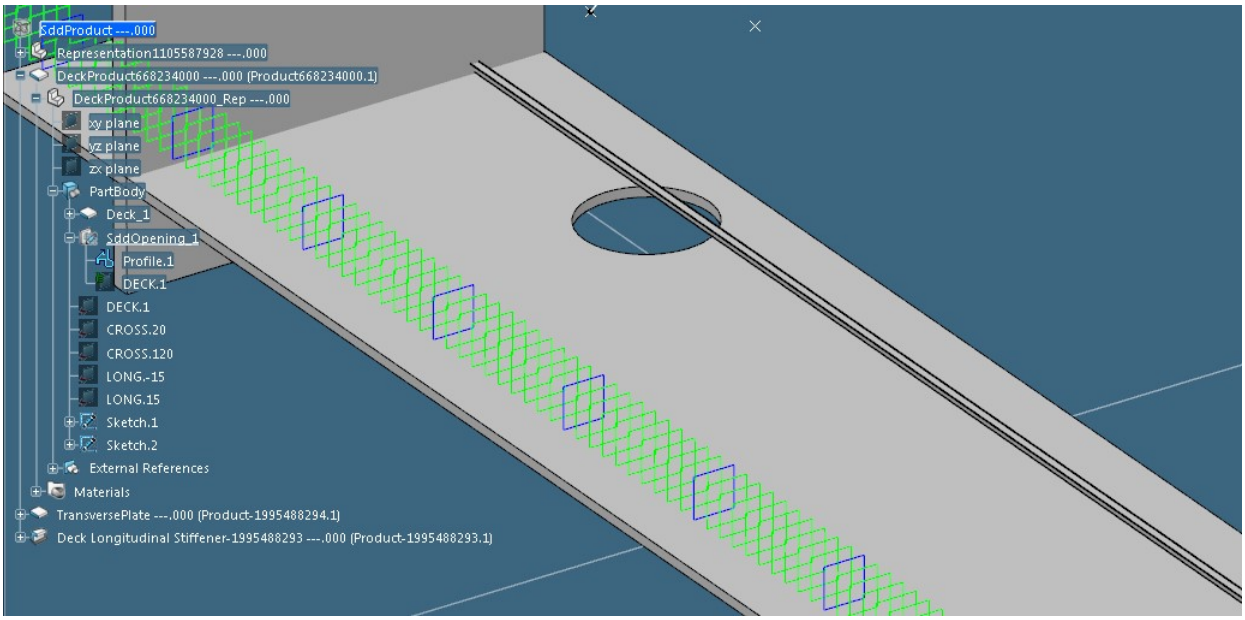
Method `SetCategory` takes opening object(iObjStrOpening) and category(iCategory) as input parameters. In this method object of type `StrCategoryMngt` is retrieved in `ObjStrCategoryMngt`. Then method `SetCategory` is called to set the catgory.

- **SetOutPutProfile method**

```
Sub SetOutPutProfile(iObjStrOpening As StrOpening, iOutputProfile As Reference, iLimitMode As Long)
'   1- Retrieves StrOpeningOutputProfile
    Dim ObjStrOpeningOutputProfile As StrOpeningOutputProfile
    Set ObjStrOpeningOutputProfile = iObjStrOpening.StrOpeningOutputProfile
'   2- Sets OutputProfile for sketch opening
    ObjStrOpeningOutputProfile.OutputProfile = iOutputProfile
'   3- Sets direction
    Set ObjRefDirection = Manager.GetReferencePlane(ObjPart, 1, "DECK.1")
    Set RefDirection = ObjPart.CreateReferenceFromObject(ObjRefDirection)
    ObjStrOpeningOutputProfile.Direction = RefDirection
'   4- Sets LimitMode
    ObjStrOpeningOutputProfile.LimitMode = iLimitMode
End Sub
```

Method `SetOutPutProfile` takes opening object(iObjStrOpening), output profile(iOutputProfile) and limit mode(iLimitMode) as input parameters. In the first step of this method object of type `StrOpeningOutputProfile` is retrieved in `ObjStrOpeningOutputProfile`. In the second step `OutputProfile` property is set to `iOutputProfile`. In the third step reference to the plane "DECK.1" is retrieved using `GetReferencePlane` method from `RfgService` and then it set as the Direction. In the fourth step it sets `LimitMode` property to iLimitMode.

Fig.1: Opening on Sketch




# Creating a Standard Opening

This use case primarily focuses on the methodology to create Standard Opening.

---

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcCreateOpeningStandard.3dxml`, `CAAScdSddUcCGR.3dxml` and `CAAScdSddUcSR.3dxml` files supplied in the folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\` where `InstallRootFolder` is the folder where the API CD-ROM is installed.

Related Topics
*<topic1>*
*<topic2>*

Where to find the macro: CAAScdSddUcCreateOpeningStandardSource.htm

This use case can be divided in six steps:

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves service of type RfgService
5. Retrieves a SDD plate object
6. Create standard Opening using Offset/Offset positioning strategy

1. **Searches and opens model which is named as "SddProduct"**

   As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim SDDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SDDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SDDPrdEditor`**, a Editor object. After searching and opening of SDD model from underlying database the current active editor returned in **`SDDPrdEditor`**.

2. **Retrieves Selection object**

   In this step UC retrieves Selection object in `SDDProdSel` variable.

```
...
  Set SDDProdSel = SDDPrdEditor.Selection
...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Dim product1Service  As  PLMProductService
  Set product1Service = DDPrdEditor.GetService("PLMProductService")
  Dim ObjVPMRootOccurrence  As  VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference  As  VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances  As  VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
```

```
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

4. **Retrieves service of type RfgService**

   In this step UC retrieves `RfgService`.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

   `GetService` method returns `RfgService`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

5. **Retrieves a SDD plate object**

   In this step UC finds a SDD plate in the part. This plate will be used as a penetrated element in the slot creation.

```
...
  Set ListOfInstances = ObjVPMReference.Instances
  Set PlateRef = ListOfInstances.Item(3).ReferenceInstanceOf
  Set PlateRepInstances = PlateRef.RepInstances
  Set PlateRepInstReference = PlateRepInstances.Item(1).ReferenceInstanceOf
  Set PlatePart = PlateRepInstReference.GetItem("Part")
  SDDProdSel.Add PlateRef
  Dim ObjSddProductPlate As SddProductPlate
  Set ObjSddProductPlate = SDDProdSel.FindObject("CATIASddProductPlate")
  Dim ObjSddPlate As SddPlate
  Set ObjSddPlate = ObjSddProductPlate.SddPlate
...
```

   In above lines, `FindObject` method finds object whose type is "CATIASddProductPlate" and returns to it. To retrieve `SddPlate` object from it, call `SddPlate` property as shown above. This will give the `SddPlate` object.

6. **Create Standard Opening with Offset/Offset positioning strategy**

   Call `CreateStandardOpeningOffsetOffset` method to create a standard opening with offset/offset positioning strategy. `CreateStandardOpeningOffsetOffse` method takes a panel object as input parameter and returns created opening as output parameter in `ObjStrOpeningOffsetOffset`.

```
...
  Dim ObjStrOpeningOffsetOffset As StrOpening
  CreateStandardOpeningOffsetOffset ObjSddPlate, ObjStrOpeningOffsetOffset
  ObjPart.Update
...
```

   The method `CreateStandardOpeningOffsetOffset` is detailed as in the below sub steps.

   i. **Retrieves all contour names and selects rectangle contour name**

```
Sub CreateStandardOpeningOffsetOffset(iObjSddPlate As SddPlate, oObjStrOpening As StrOpening)
    Dim ContourNames() As Variant
    GetContourNames iObjSddPlate, ContourNames
    Dim ContourName As String
    ContourName = ContourNames(1)
...
```

   In this step Uc retrieves all the contour names available, by calling method `GetContourNames`. This method takes a panel as input parameter and returns the names in an array as output parameter. Then Rectangle contour name which is at index 1 is stored in `ContourName` variable.

   The method `GetContourNames` is detailed as in the below sub steps.

   i. 
```
Sub GetContourNames(iObjSddPlate As SddPlate, oContourNames() As Variant)
    'Get StrOpeningsMgr object
    Dim ObjStrOpeningsMgr As StrOpeningsMgr
    Set ObjStrOpeningsMgr = iObjSddPlate.StrOpeningsMgr
    'Get available contour names
    ObjStrOpeningsMgr.GetAvailableStandardContours oContourNames
End Sub
```

   In this method object `ObjStrOpeningsMgr` of type `StrOpeningsMgr` is retrieved from iObjSddPlate. Then method `GetAvailableStandardContours` called to get the contour names.

   ii. **Retrieves contour params**

```
...
    Dim StdContourParms As StrStandardContourParameters
    GetContourParams iObjSddPlate, ContourName, StdContourParms
...
```

   In this step Uc retrieves the parameters of a particular contour. To do this Uc calls a method `GetContourParams`. This method takes a panel object and name the contour as input parameter and it returns the collection of contour parameters as output parameter in `StdContourParms`. Here `StdContourParms` is a collection object of type `StrStandardContourParameters`, which is a collection of `StrParameter`.

   The method `GetContourParams` is detailed as in the below sub steps.

   i. 
```
Sub GetContourParams(iObjSddPlate As SddPlate, iContourName As String, oContourParms As StrStandardContourParameters)
    'Get StrOpeningsMgr object
    Dim ObjStrOpeningsMgr As StrOpeningsMgr
    Set ObjStrOpeningsMgr = iObjSddPlate.StrOpeningsMgr

    'Get parameters of contour
    Set oContourParms = ObjStrOpeningsMgr.GetStandardContourParms(iContourName)
    If oContourParms.Count = 0 Then
        Err.Raise 1, Err.Source, "No contour params found"
        Exit Sub
    End If
```

```
End Sub
```

In this method object `ObjStrOpeningsMgr` of type `StrOpeningsMgr` is retrieved from iObjSddPlate. Then method `GetStandardContourParms` is cal to get the contour parameters. Method `GetStandardContourParms` takes contour name as input parameter. Contour parameters related to this name w be returned in `oContourParms`.

iii. **Sets contour parameter's data**

```
...
    SetRectContourParamsData StdContourParms, "1000mm", "400mm", "50mm"
...
```

In this step Uc sets the data of contour parameters by calling method `SetRectContourParamsData`. Method `SetRectContourParamsData` takes contour parameters collection object, width, height and corner radius as input parameters.

iv. **Retrieves all standard positiong strategy names and selects offset offset positioning strategy**

```
...
    Dim StdPosStrategyNames() As Variant
    GetStdPosStrategyNames iObjSddPlate, StdPosStrategyNames
    'Select Positioning strategy offset offset
    Dim PosStratName As String
    PosStratName = StdPosStrategyNames(0)
...
```

In this step Uc retrieves all the standard positioning strategy names available, by calling method `GetStdPosStrategyNames`. This method takes a SDD plate input parameter and returns the names in an array as output parameter. Then "Offset/Offset" contour name which is at index 0 is stored in `PosStratName` variable.

The method `GetStdPosStrategyNames` is detailed as in the below sub steps.

i.
```
Sub GetStdPosStrategyNames(iObjSddPlate As SddPlate, oStdPosStrategyNames() As Variant)
    'Get StrOpeningsMgr object
    Dim ObjStrOpeningsMgr As StrOpeningsMgr
    Set ObjStrOpeningsMgr = iObjSddPlate.StrOpeningsMgr
    'Get available standard positioning strategy names
    ObjStrOpeningsMgr.GetAvailableStandardPositioningStrategies oStdPosStrategyNames
End Sub
```

In this method object `ObjStrOpeningsMgr` of type `StrOpeningsMgr` is retrieved from iObjSddPlate. Then method `GetAvailableStandardPositioningStrategies` is called to get the standard positioning strategy names.

v. **Retrieves standard positioning strategy parameters**

```
...
    Dim PosStratParms As StrStandardPosStrategyParameters
    GetStdPosStrategyParams iObjSddPlate, PosStratName, PosStratParms
...
```

In this step Uc retrieves the parameters of a particular standard positioning strategy. To do this Uc calls a method `GetStdPosStrategyParams`. This method takes a SDD plate object and name of the standard positioning strategy as input parameter and it returns the collection of standard positioning strategy parameters as output parameter in `PosStratParms`. `PosStratParms` is a collection object of type `StrStandardPosStrategyParameters`, which is collectio parameters required for defining positioning strategy.

The method `GetStdPosStrategyParams` is detailed as in the below sub steps.

i.
```
Sub GetStdPosStrategyParams(iObjSddPlate As SddPlate, iStdPosStrategyName As String,
                            oStdPosStrategyParms As StrStandardPosStrategyParameters)
    'Get StrOpeningsMgr object
    Dim ObjStrOpeningsMgr As StrOpeningsMgr
    Set ObjStrOpeningsMgr = iObjSddPlate.StrOpeningsMgr

    'Get standard positioining strategy parameters
    Set oStdPosStrategyParms = ObjStrOpeningsMgr.GetStandardPositioningStrategyParms(iStdPosStrategyName)
    If oStdPosStrategyParms.Count = 0 Then
        Err.Raise 1, Err.Source, "Positioning strategy parameters cannot found"
        Exit Sub
    End If
End Sub
```

In this method object `ObjStrOpeningsMgr` of type `StrOpeningsMgr` is retrieved from iObjSddPlate. Then method `GetStandardPositioningStrategyParms` is called to get the positioning strategy parameters. Method `GetStandardPositioningStrategyParms` ta positioning strategy name as input parameter. Positioning strategy parameters related to this name will be returned in `oStdPosStrategyParms`.

vi. **Sets standard positioning strategy parameter's data**

```
...
    SetStdPosStrategyParamsDataForOffsetOffset PosStratParms
...
```

In this step Uc sets the data of "offset/offset" positioning strategy parameters by calling method `SetStdPosStrategyParamsDataForOffsetOffset`. Metho `SetStdPosStrategyParamsDataForOffsetOffset` takes positioning strategy parameters collection object(`PosStratParms`) as input parameters.

vii. **Create an opening and set its type, category, standard opening parameters**

i. **Create opening with no parameters**

```
...
    'AddOpening with no parameters
    AddOpening iObjSddPlate, oObjStrOpening
...
```

In this step Uc an opening with no parameters and stores in `oObjStrOpening` variable.

The method `AddOpening` is detailed as in the below sub steps.

```
Sub AddOpening(iObjSddPlate As SddPlate, oObjStrOpening As StrOpening)
    'Get StrOpenings object
    Dim ObjStrOpenings As StrOpenings
    Set ObjStrOpenings = iObjSddPlate.StrOpenings
    'Add Opening
    Set oObjStrOpening.= ObjStrOpenings.Add
End Sub
```

In this method object `ObjStrOpenings` of type `StrOpenings` is retrieved from iObjSddPlate. Then method `Add` is called to create an opening with no parameters.

ii. **Set Opening type**

```
...
    'set opening type catStrOpeningModeStandard for standard opening
    SetOpeningType oObjStrOpening, catStrOpeningModeStandard
...
```

In this step Uc sets the type of opening as catStrOpeningModeStandard which is standard opening type.

The method `SetOpeningType` is detailed as in the below sub steps.

```
Sub SetOpeningType(iObjStrOpening As StrOpening, iOpeningType As Long)
    'Set Opening type
    Set iObjStrOpening.OpeningType = iOpeningType
End Sub
```

In this method, property `OpeningType` is used to set the opening type of the opening.

iii. **Set Opening category**

```
...
    'set category
    SetCategory oObjStrOpening
...
```

In this step Uc sets the category of the opening.

The method `SetCategory` is detailed as in the below sub steps.

```
Sub SetCategory(iObjStrOpening As StrOpening)
    'Get StrCategoryMngt object
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = iObjStrOpening.StrCategoryMngt
    'Set Opening Category
    Set ObjStrCategoryMngt.SetCategory "CATSfmOpeningExt"
End Sub
```

In this method object `ObjStrCategoryMngt` of type `StrCategoryMngt` is retrieved from iObjStrOpening. Then method `SetCategory` is used to set th category of the opening.

iv. **Set standard opening parameters**

```
...
    SetStandardOpeningParameters oObjStrOpening, ContourName, StdContourParms, PosStratName, PosStratParms
...
```

In this step Uc sets some of the data of the created opening `oObjStrOpening`. First parameter is opening created previously Second parameter is conte name and Third parameter is contour parameters Fourth parameter is positioning strategy name and fifth parameter is positioing strategy parameters.

viii. **Updates created opening object**

```
...
    ObjPart.UpdateObject oObjStrOpening
End Sub
```

Method `UpdateObject` updates the created opening.

The source also includes standard opening creation using positioning strategies Half Height/Offset, Mid Dist/Offset, Mid Dist/Mid Dist, Half Height/Mid D To see the source go to CAAScdSddUcCreateOpeningStandardSource.htm.

**Detailed steps of methods called in Uc**

- **SetRectContourParamsData method**

This method sets width, height and corner radius for Rectangle type of contour. This method takes collection of contour parameters, width, height and corner radiu input parameters.

1. **Retrieves total number of contour parameters**

```
Sub SetRectContourParamsData(iContourParms As StrStandardContourParameters, iStrW As String, iStrH As String, iStrCR As Strin
    Dim NbOfContourParams As Long
    NbOfContourParams = iContourParms.Count
...
```

Here size of the of the contour parameters is retrieved.

2. **Retrieves a parameter from collection and set width/height/CornerRadius**

```
...
```

```
    For i = 1 To NbOfContourParams
      'Get a StrParameter from the collection
      Dim ObjStrParameter As StrParameter
      Set ObjStrParameter = iContourParms.Item.(i)
      'Get role of the parameter
      StrRole = ObjStrParameter.Role
      'Get parameter from StrParameter
      Dim ContourParam As Parameter
      Set ContourParam = ObjStrParameter.Parameter

      'Sets width
      If StrRole = "Str_Width" Then
        ContourParam.ValuateFromString (iStrW)
      End If

      'Sets height
      If StrRole = "Str_Height" Then
        ContourParam.ValuateFromString (iStrH)
      End If

      'Sets corner radius
      If StrRole = "Str_CornerRadius" Then
        ContourParam.ValuateFromString (iStrCR)
      End If
    Next
End Sub
```

Here collection object `iContourParms` which is of type `StrStandardContourParameters` is collection of `StrParameter`. So here in for loop Uc retrieves a StrParameter then its role and parameter is retrieved and by checking its role(width/height/CornerRadius) its value is set.

- **SetStdPosStrategyParamsDataForOffsetOffset method**

This method sets U, V references and angle parameter value for "offset/offset" positioning strategy. This method takes collection of positioning strategy parameter (`StrStandardPosStrategyParameters`) as input parameters.

1. **Retrieves the number of Standard Positioning Strategy Parameters**

```
Sub SetStdPosStrategyParamsDataForOffsetOffset(iStdPosStrategyParms As StrStandardPosStrategyParameters)
    Dim SizeOfStdPosStratParms As Long
    SizeOfStdPosStratParms = iStdPosStrategyParms.Count
...
```

Here size of the collection of positioning strategy parameters is retrieved.

2. **Retrieves a positioning strategy parameter from the list**

```
...
    For i = 1 To SizeOfStdPosStratParms
        Set ObjStrStdPosStParam = iStdPosStrategyParms.Item(i)
...
```

Here a For loop is running through the collection of positioing strategy parameters. A positioning strategy parameter is retrieved in `ObjStrStdPosStParam`.

3. **Sets U and V references**

```
...
        If (TypeName(ObjStrStdPosStParam) = "StrRefOffset") Then
            'Retrieves role of the parameter
            StrRole = ObjStrStdPosStParam.Role
            'Sets reference for U
            If StrRole = "PosSpecUCurve" Then
              Set ObjRefUShift = Manager.GetReferencePlane(ObjPart, 3, "LONG.-7")
              Set RefUShift = ObjPart.CreateReferenceFromObject(ObjRefUShift)
              ObjStrStdPosStParam.SetSpecification Nothing, RefUShift
              'set relevant side to Port
              ObjStrStdPosStParam.SetRelevantSide (4)
            End If
            'Sets reference for V
            If StrRole = "PosSpecVCurve" Then
              Set ObjRefVShift = Manager.GetReferencePlane(ObjPart, 2, "CROSS.94")
              Set RefVShift = ObjPart.CreateReferenceFromObject(ObjRefVShift)
              ObjStrStdPosStParam.SetSpecification Nothing, RefVShift
              'set relevant side to Fore
              ObjStrStdPosStParam.SetRelevantSide (2)
              'Set offset
              Set OffsetParm = ObjStrStdPosStParam.GetOffsetParm
              OffsetParm.ValuateFromString ("1000mm")
            End If
        End If
...
```

Here if parameter is of type `StrRefOffset` then U and V references are set. First role of the parameter is retrieved. If the role is of type **"PosSpecUCurve"** it is U reference. To set the U reference, first the reference to the plane LONG.-7 is created and then it is set by using method `SetSpecification`. After setti the U reference its side is set to Port side by calling method `SetRelevantSide(4)`. Here 4 means the port side. If the role is of type **"PosSpecVCurve"** then is V reference. To set the V reference, first the reference to the plane CROSS.94 is created and then it is set by using method `SetSpecification`. After setti the V reference its side is set to Fore side by calling method `SetRelevantSide(2)`. Here 2 means the Fore side. To set the offset for the V reference, first of parameter is retrieved by calling method `GetOffsetParm` then its value set to 1000mm.

4. **Sets Angle parameter's value**

```
...
        If (TypeName(ObjStrStdPosStParam) = "StrPosAxisAdjustment") Then
            'get angle parameter and set its value
            Dim AngleParm As Parameter
            Set AngleParm = ObjStrStdPosStParam.GetAngleParameter
            AngleParm.ValuateFromString ("45deg")
        End If
    Next
End Sub
```

To set the angle first angle parameter is retrieved by using method `GetAngleParameter`. Then its value is set to 45deg.

- **SetStandardOpeningParameters method**

  This method sets contour name, contour parameters, positioning strategy name and positioning strategy parameters of the opening

  1. **Retrieves StrOpeningStandard and sets Direction, LimitMode and contour and positiong strategy parameters**

```
...
    'Get StrOpeningStandard object
    Dim ObjStrOpeningStandard As StrOpeningStandard
    Set ObjStrOpeningStandard = oObjStrOpening.StrOpeningStandard
    'set direction
    Set ObjDirectionRef = Manager.GetReferencePlane(ObjPart, 1, "DECK.6")
    Set DirectionRef = ObjPart.CreateReferenceFromObject(ObjDirectionRef)
    ObjStrOpeningStandard.Direction = DirectionRef
    'set limit mode
    ObjStrOpeningStandard.LimitMode = 0
    'set contour and positiong strategy parameters
    ObjStrOpeningStandard.SetContourAndPosStrategy iContourName, iStdContourParms, iPosStratName, iPosStratParms
End Sub
```

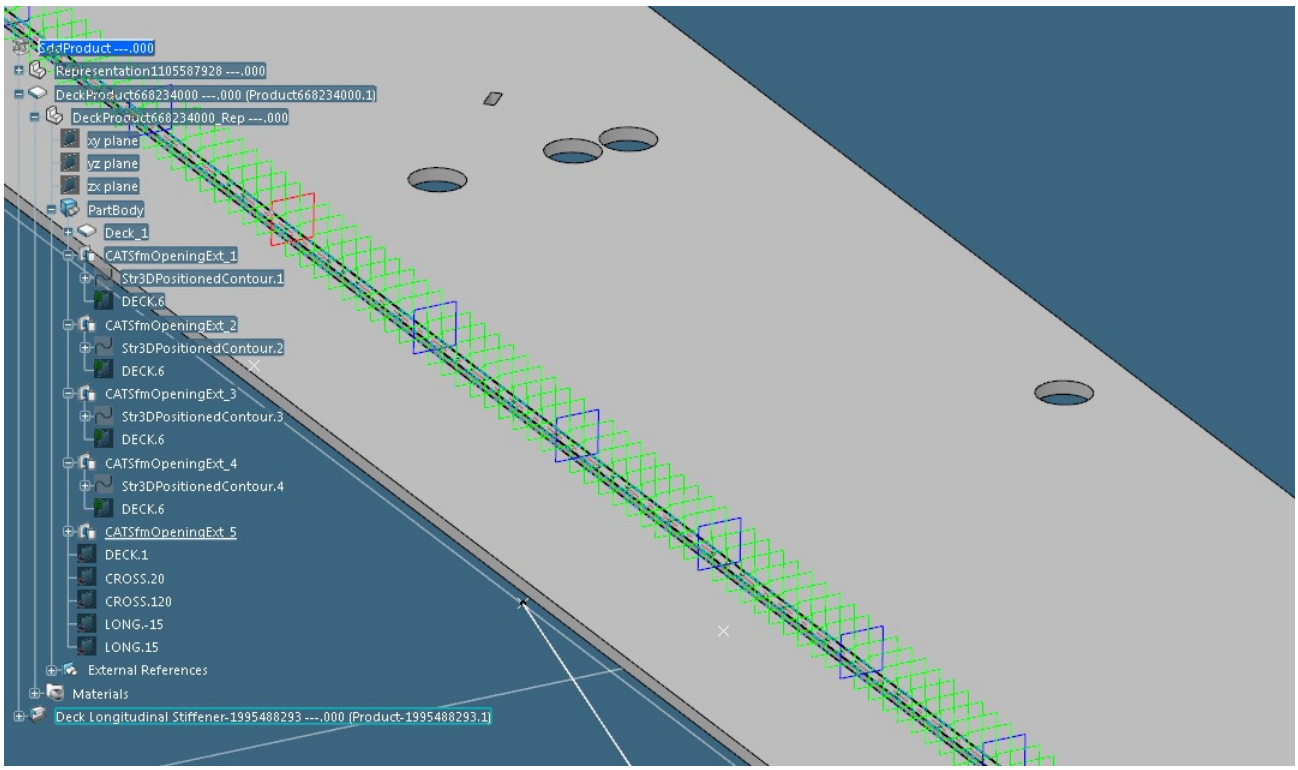
  In this step object `ObjStrOpeningStandard` of type `StrOpeningStandard` is retrieved. Then reference to the DECK.6 plane is retrieved and it set as `Direction`. `LimitMode` property is set to 0. Here 0 means UptoLast. Then to set the contour name contour parameters, positioning strategy name and positioning strategy parameters method `SetContourAndPosStrategy` is called on object `ObjStrOpeningStandard`.

  2. **Retrieves StrOpeningExtrusionMngt and sets forming extrusion mode**

```
...
    Set ObjStrOpeningExtrusionMngt = oObjStrOpening.StrOpeningExtrusionMngt
    ObjStrOpeningExtrusionMngt.ExtrusionMode = 1
...
```

  In this step object forming mode is set. To set the forming mode, forming mode value is assigned to property `ExtrusionMode`. Here 1 defines Before Formi Extrusion Mode..

Fig.1: Standerd Opening




# Creating an SDD Endcut

This use case primarily focuses on the methodology to create a SDD Endcut.

---

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcCGR.3dxml`, `CAAScdSddUcSR.3dxml`, `CAAScdSddUcCreateEndCut.3dxml` files supplied in folder `InstallRootFolder\CAADoc\English\CAAScdSddSDDesign\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcCreateEndcutSource.htm

This use case can be divided in six steps:

Related Topics
Structure Design Object Model Map
Launching an Automation Use Case

1. **Searches and opens model which is named as "SddProduct"**

   As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim SDDPrdEditor As Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SDDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SDDPrdEditor`**, a Editor object. After searching and opening of SDD model from underlying database the current active editor is returned in **`SDDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SDDProdSel variable. To retrieve the Selection object **`SDDPrdEditor`** is used.

```
 ...
    Set SDDProdSel = SDDPrdEditor.Selection
 ...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Dim product1Service As PLMProductService
  Set product1Service = SDDPrdEditor.GetService("PLMProductService")
  Dim ObjVPMRootOccurrence As VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference As VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances As VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

4. **Retrieves a SDD product Stiffener object**

   In this step UC retrieves product stiffener object. Endcut will be created on this stiffener.

```
...
  Set ListOfInstances = ObjVPMReference.Instances
  Set StiffenerRef = ListOfInstances.Item(2).ReferenceInstanceOf
  SDDProdSel.Add StiffenerRef
  Set ObjSddProductStiffener = SDDProdSel.FindObject("CATIASddProductStiffener")
...
```

   `FindObjectByName` method finds object whose name is "CATIASddProductStiffener" and returns reference to it.

5. **Creates Endcut**

   Now, stiffener product is retrieved. Call `CreateEndCut` method to create Endcut on the stiffener in the stiffener product. `CreateEndCut` method takes a product stiffener as input parameter and it returns created Endcut as output parameter.

```
...
  Dim ObjStrEndcut As StrEndcut
  CreateEndCut ObjSddProductStiffener, ObjStrEndcut
...
```

   The method `CreateEndCut` is detailed as in the below sub steps.

   i. **Retrieves `StrEndcutMngt` object and and create an Endcut with no data.**

```
Sub CreateEndCut(iObjProductSfdStiffener As SfdProductStiffener, oObjStrEndCut As StrEndCut)
  'Get StrEndcutMngt object
  Dim ObjSddStiffener As SddStiffener
  Set ObjSddStiffener = iObjSddProductStiffener.SddStiffener
  Dim ObjStrEndcutMngt As StrEndcutMngt
  Set ObjStrEndcutMngt = iObjSfdStiffener.StrEndcutMngt
  'Create Endcut
  Set oObjStrEndCut = ObjStrEndcutMngt.AddEndCut(2)
...
```

   `StrEndcutMngt` is retrieved in `ObjStrEndcutMngt` variable from `ObjSddStiffener` object as shown above. On `ObjStrEndcutMngt` object, `AddEndCut` method is called to create the Endcut with no data. Method `AddEndCut` takes one input parameter which defines whether this Endcut is created at the start of the profile or end of the profile. 1 is for start and 2 is for end. Now Uc needs to set the different properties of the Endcut like Endcut's parameters etc.

   ii. **Sets Endcut parameters**

```
...
  SetSNIPE_WT_radiusEndCutParameters oObjStrEndCut
...
```

   Method `SetSNIPE_WT_radiusEndCutParameters` is called to set the Endcut parameters.

   The method `SetSNIPE_WT_radiusEndCutParameters` is detailed as in the below sub steps.

1. **Retrieves `StrDetailFeature` object**

```
Sub SetSNIPE_WT_radiusEndCutParameters(iObjStrEndCut As StrEndCut)
  Dim ObjStrDetailFeature As StrDetailFeature
  Set ObjStrDetailFeature = iObjStrEndCut.StrDetailFeature
...
```

`ObjStrDetailFeature` is of type `StrDetailFeature`. It is retrieved from `iObjStrEndCut` object using method `StrDetailFeature`. Later this object is used to set different Endcut parameters like Endcut name, Endcut type etc.

2. **Sets Endcut name**

```
...
  ObjStrDetailFeature.InitByName "WT_snipe_radius"
...
```

`InitByName` method of `StrDetailFeature` is called with parameter "WT_snipe_radius". This defines the name of the Endcut.

3. **Retrieves list of parameters of EndCut**

```
...
  Dim EndcutParameters As StrParameters
  Set EndcutParameters = ObjStrDetailFeature.GetParameters
...
```

`GetParameters` method of `ObjStrDetailFeature` returns list of parameters of `ObjStrEndCut`

4. **Sets radius parameter of end cut**

```
...
  Dim Radius As Parameter
  Set Radius = EndcutParameters.Item(1)
  Radius.ValuateFromString ("70mm")...
```

`Radius` parameter `EndcutParameters` is retrieved from the list of parameters and its value is set.

5. **Updates the ObjStrDetailFeature object**

```
...
  ObjStrDetailFeature.Update
End Sub
```

Call to `Update` method of `ObjStrDetailFeature` will update the Endcut feature.

iii. **Updates the Part object**

```
...
  ObjPart.Update
End Sub
```

`Update` method updates the `ObjPart` Part object

# Creating a SDD Slot

This use case primarily focuses on the methodology to create a SDD slot.

Related Topics
Structure Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcCreateSlot.3dxml`, `CAAScdSddUcCGR.3dxml`, `CAAScdSddUcSlotSections.3dxml` and `CAAScdSddUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcCreateSlotSource.htm

This use case can be divided in 7 steps

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves a SDD stiffener object
5. Retrieves a SDD plate object
6. Creates Slot
7. Updates the Part object

1. **Searches and opens model which is named as "SddProduct"**

As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim SDDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SDDPrdEditor
...
```

The function **`OpenProduct`** returns **`SDDPrdEditor`**, a Editor object. After searching and opening of SDD model from underlying database the current active editor is returned in **`SDDPrdEditor`**.

2. **Retrieves Selection Object**

As a next step, the UC retrieves Selection object in SDDProdSel variable. To retrieve the Selection object **`SDDPrdEditor`** is used.

```
...
   Set SDDProdSel = SDDPrdEditor.Selection
...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Dim product1Service  As  PLMProductService
  Set product1Service = SDDPrdEditor.GetService("PLMProductService")
  Dim ObjVPMRootOccurrence  As  VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference  As  VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances  As  VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

4. **Retrieves a SDD Stiffener object**

   In this step UC retrieves stiffener object. This stiffener will be used as a penetrating element for the slot creation.

```
...
  Set ListOfInstances = ObjVPMReference.Instances
  Set StiffenerRef = ListOfInstances.Item(2).ReferenceInstanceOf
  Set StiffenerRepInstances = StiffenerRef.RepInstances
  Set StiffenerRepInstReference = StiffenerRepInstances.Item(1).ReferenceInstanceOf
  Set StiffenerPart = StiffenerRepInstReference.GetItem("Part")
  Dim ObjSddProductStiffener  As  SddProductStiffener
  SDDProdSel.Add StiffenerRef
  Set ObjSddProductStiffener = SDDProdSel.FindObject("CATIASddProductStiffener")
  Dim ObjSddStiffener  As  SddStiffener
  Set ObjSddStiffener = ObjSddProductStiffener.SddStiffener

...
```

`FindObject` method finds object whose type is "CATIASddProductStiffener" and returns it. To retrieve `SddStiffener` object from `ObjSddProductStiffener` call `SddStiffener` property as shown above. This will give the `SddStiffener` object. .

5. **Retrieves a SDD plate object**

   In this step UC finds a SDD plate in the part. This plate will be used as a penetrated element in the slot creation.

```
...
  Set ListOfInstances = ObjVPMReference.Instances
  Set PlateRef = ListOfInstances.Item(3).ReferenceInstanceOf
  Set PlateRepInstances = PlateRef.RepInstances
  Set PlateRepInstReference = PlateRepInstances.Item(1).ReferenceInstanceOf
  Set PlatePart = PlateRepInstReference.GetItem("Part")
  SFDProdSel.Add PlateRef
  Dim ObjSddProductPlate As SddProductPlate
  Set ObjSddProductPlate = SDDProdSel.FindObject("CATIASddProductPlate")
  Dim ObjSddPlate As SddPlate
  Set ObjSddPlate = ObjSddProductPlate.SddPlate
...
```

In above lines, `FindObject` method finds object whose type is "CATIASddProductPlate" and returns to it. To retrieve `SddPlate` object from it, call `SddPlate` property as shown above. This will give the `SddPlate` object.

6. **Creates Slot**

   Now, penetrated element (plate) and penetrating element (stiffener) are available to create slot. Call `CreateSlot` method to create slot. `CreateSlot` method takes a plate and a stiffener as input parameters and it returns created slot as output parameter.

```
...
  Dim ObjStrSlot As StrSlot
  CreateSlot ObjSfdPanel, ObjSfdStiffener, ObjStrSlot
...
```

The method `CreateSlot` is detailed as in the below sub steps.

   i. **Retrieves `StrSlots` object and and create slot with no data.**

```
Sub CreateSlot(iObjSddPlate As SddPlate, iObjSddStiffener As SddStiffener, oObjStrSlot As StrSlot)
'Get StrSlots object
    Dim ObjStrSlots As StrSlots
    Set ObjStrSlots = iObjSddPlate.StrSlots
'Create StrSlot
    Set oObjStrSlot = ObjStrSlots.Add
...
```

`StrSlots` is retrieved in `ObjStrSlots` variable from `iObjSddPlate` object. On `ObjStrSlots` object, `Add` method is called to create the slot with no data. Now Uc needs to set the different properties of the slot like penetrating element, slot parameters etc.

   ii. **Sets penetrating profile for slot**

```
...
'set penetrating element
  Dim penetratingElem As Reference
  Set penetratingElem = StiffenerPart.CreateReferenceFromObject(iObjSddStiffener)
    oObjStrSlot.SetPenetratingProfile penetratingElem
...
```

`penetratingElem` is of type `Reference`. Method `CreateReferenceFromObject` creates reference to `iObjSddStiffener` and returns it in `penetratingElem`. Then `SetPenetratingProfile` method is called to set the penetrating profile for the slot.

iii. **Sets slot parameters**

```
...
    SetSlotParameters oObjStrSlot, iObjSddPlate
...
```

Method `SetSlotParameters` is called to set the slot parameters of respective slot which is set.

The method `SetSlotParameters` is detailed as in the below sub steps.

i. **Retrieves `StrDetailFeature` object**

```
Sub SetSlotParameters(iObjStrSlot As StrSlot,iObjSddPlate As SddPlate)
    Dim ObjStrDetailFeature As StrDetailFeature
    Set ObjStrDetailFeature = iObjStrSlot.StrDetailFeature
...
```

`ObjStrDetailFeature` is of type `StrDetailFeature`. It is retrieved from `iObjStrSlot` object using property `StrDetailFeature`. Later this object is used to set different slot parameters like slot name, slot type etc.

ii. **Sets Slot type**

```
...
    ObjStrDetailFeature.Type = "RECT"
...
```

`Type` property of `SfdDetailFeature` is set to "RECT". This defines the type of the slot.

iii. **Sets Slot name**

```
...
    ObjStrDetailFeature.FeatureName = "RECT_UNI_WT"
...
```

`FeatureName` property of `SfdDetailFeature` is set to "RECT_UNI_WT". This defines the name of the slot.

iv. **Retrieves list of parameters of impacted object**

```
...
    Set ObjPartParameters = PlatePart.Parameters
    'parameters will be aggregated under impacted
    'object(here impacted object is iObjSddPlate)
    Set HBParameters = ObjPartParameters.SubList(iObjSddPlate, True)
...
```

`Parameters` property of `PlatePart` returns collection object containing the part parameters. `SubList` method from `Parameters` returns the sub-collection of parameters aggregated to `iObjSddPlate` object and it collected in `HBParameters` object.

v. **Creates parameters and stores role of each parameter in an array**

```
...
    Set DBB = HBParameters.CreateDimension("DBB", "LENGTH", "300")
    Set DR = HBParameters.CreateDimension("DR", "LENGTH", "200")
    Set DL = HBParameters.CreateDimension("DL", "LENGTH", "200")
    Set DTT = HBParameters.CreateDimension("DTT", "LENGTH", "50")
    'Define parameter roles in an array
    Dim ParamRoles(3) As Variant
    ParamRoles(0) = "DBB"
    ParamRoles(1) = "DR"
    ParamRoles(2) = "DL"
    ParamRoles(3) = "DTT"
...
```

Method `CreateDimension` creates a user dimension. Here Uc creates 4 parameters (DBB, DR, DL, DLL) for slot. Role of each parameter is stored in ParamRoles array.

vi. **Sets parameters**

```
...
    ObjStrDetailFeature.SetParameters HBParameters, ParamRoles
...
```

`SetParameters` method of `ObjStrDetailFeature` is called to set the parameters for slot creation. This method takes two input parameters. First parameter is the collection object of slot parameters and second object is an array which contains the role of each parameter from the collection object.

vii. **Updates the ObjStrDetailFeature object**

```
...
    ObjStrDetailFeature.Update
End Sub
```

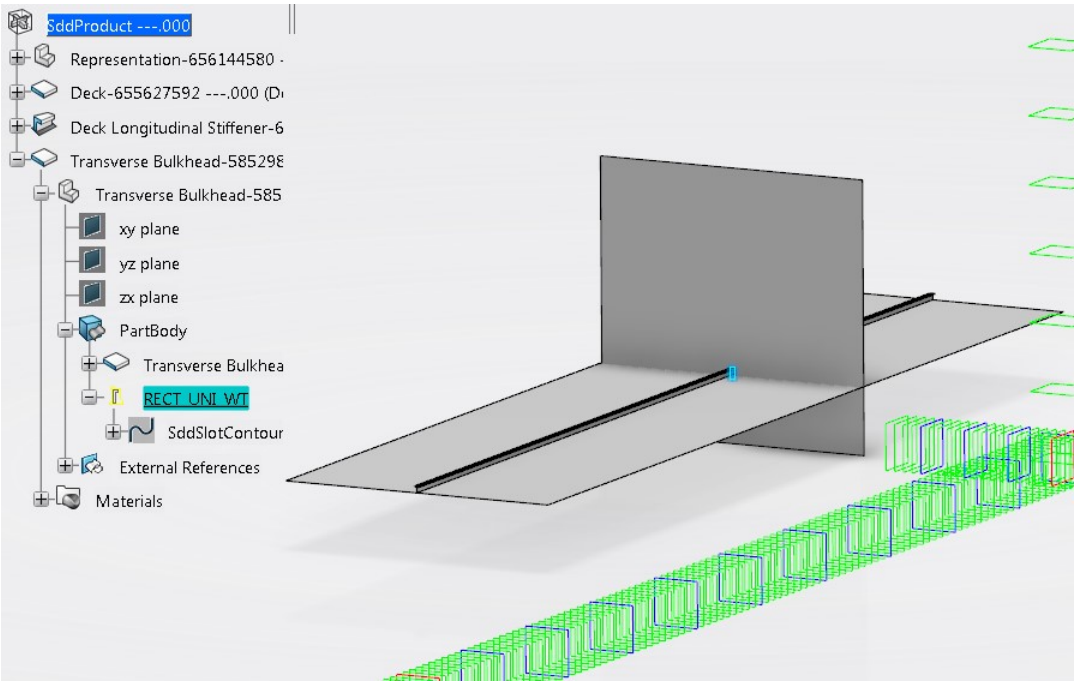
Call to `Update` method of `ObjStrDetailFeature` will update connection coordinate and visualization of the slot feature.

7. **Updates the product plate object**

```
...
  ObjSddProductPlate.Update
End Sub
```

`Update` method updates the `ObjSddProductPlate`

Fig.1: Slot




# Creating an SDD Stiffener on Free Edge

This use case primarily focuses on the methodology to create a SDD stiffener On Free Edge.

Related Topics
Structure Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcCreateStiffenerOnFreeEdge.3dxml`, `CAAScdSddUcWT18x179_5.3dxml`, `CAAScdSddUcCGR.3dxml` and `CAAScdSddUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcCreateStiffenerOnFreeEdgeSource.htm

This use case can be divided in nine steps:

1. Searches and opens model which is named as "SDDProduct"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves a SddPlate
5. Retrieves SddStiffenerMngt
6. Creates a Stiffener On Free Edge on a Plate Limit
7. Creates a Stiffener On Free Edge on an opening on the Plate

1. **Searches and opens model which is named as "SDDProduct"**

   As a first step, the UC retrieves a model "SDDProduct" from DB and loads it and returns object of the Editor.

```
...
  Dim SDDPrdEditor  As  Editor
  Dim prdName As String
  prdName = "SddProduct"
  OpenProduct prdName , SDDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SDDPrdEditor`**, a Editor object. After searching and opening of SDD model from underlying database the current active editor returned in **`SDDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves the representation variable. To retrieve the Selection object **`SDDPrdEditor`** is used.

```
 ...
  Set SDDProdSel = SDDPrdEditor.Selection
...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Dim product1Service As PLMProductService
  Set product1Service = SDDPrdEditor.GetService("PLMProductService")
  Dim ObjVPMRootOccurrence As VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference As VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances As VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
```

```
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

4. **Retrieves a SddPlate object**

   In this step UC retrieves a SddPlate from SddProductPlate which is retrieved using Selection object.

```
...
  Set PlateRef = ListOfInstances.Item(2).ReferenceInstanceOf
  SDDProdSel.Add PlateRef
  Set ObjSddProductPlate1 = SDDProdSel.FindObject("CATIASddProductPlate")
  Set ObjSddPlate1 = ObjSddProductPlate1.SddPlate
...
```

   In above lines, *FindObject* method finds the object "CATIASddProductPlate" and returns it to "ObjSddProductPlate1". To retrieve `SddPlate` object, call *SddPlate*
   as shown above. This will give the `SddPlate` object.

5. **Retrieves a SddStiffenerMngt object**

   In this step UC retrieves a SddStiffenerMngt object using Selection object.

```
...
  Dim ObjSddStiffenerMngt As SddStiffenerMngt
  SDDProdSel.Add ObjVPMRootOccurrence
  Set ObjSddStiffenerMngt = SDDProdSel.FindObject("CATIASddStiffenerMngt")
...
```

   In above lines, *FindObject* method finds the object "CATIASddStiffenerMngt" and returns it to "ObjSddStiffenerMngt". This will give the `SddStiffenerMngt` ob

6. **Creates a Stiffener On Free Edge on a Plate Limit**

   Now plate is available to create stiffener on it. Call `CreateStiffenerOnFreeEdge_PlateLimit` method to create stiffener. Inputs to
   `CreateStiffenerOnFreeEdge_PlateLimit` method are a SddStiffenerMngt object and an empty SddProductStiffenerOnFreeEdge object. When the function exe
   `ObjSddProductStiffenerOnFreeEdge_PlateLimit` stores the Stiffener On Free Edge object created on on of the Limits of the Plate.

```
...
  Dim ObjSddProductStiffenerOnFreeEdge_PlateLimit As SddProductStiffenerOnFreeEdge
  CreateStiffenerOnFreeEdge_PlateLimit ObjSddStiffenerMngt, ObjSddProductStiffenerOnFreeEdge_PlateLimit
...
```

   The method `CreateStiffenerOnFreeEdge_PlateLimit` is detailed as in the below sub steps.

```
...
Sub CreateStiffenerOnFreeEdge_PlateLimit(iObjSddStiffenerMngt As SddStiffenerMngt, oObjProdStiffener As SddProductStiffenerOnFreeE
...
```

   i. Create a new empty stiffener by the method `AddStiffenerOnFreeEdge` from object `SddStiffenerMngt`

```
...
  Sub CreateStiffenerOnFreeEdge(iObjSddStiffenerMngt As SddStiffenerMngt, oObjProductStiffener As SddProductStiffenerOnFreeEd
    ' Add Stiffener On Free Edge
    Set oObjProductStiffener = iObjSddStiffenerMngt.AddStiffenerOnFreeEdge
  End Sub
...
```

   On `iObjSddStiffenerMngt` object `AddStiffenerOnFreeEdge` method is called to create the empty stiffener. Now UC needs to set the different properties
   stiffener like material, category etc.

   ii. Set different properties of the stiffener on free edge like material. Withing the script, we call the subroutine SetMaterial which Retrieve the `StrMaterialMn`
   from `SddProductStiffenerOnFreeEdge` object and set the material for the stiffener using `SetMaterial` method on the `StrMaterialMngt` object. Below w
   what is done within the subroutine. Although, this could be done within the `CreateStiffenerOnFreeEdge` subroutine, we are using a separate `SetMateria`
   can use it when we create a `StrSddStiffenerOnFreeEdge` on an Opening and an Extracted Edge.
   Here is what is done within the SetMaterial method:

```
...
Sub SetMaterial(iObjProductStiffener As SddProductStiffenerOnFreeEdge, iStrMaterial As String)
    ' Get StrMaterialMngt object
    Dim ObjMaterialMngt As StrMaterialMngt
    Set ObjMaterialMngt = iObjProductStiffener.StrMaterialMngt
    ' Set material of the stiffener
    ObjMaterialMngt.SetMaterial iStrMaterial
End Sub
...
```

   A `SddProductStiffenerOnFreeEdge` and a `String` indicating the name of the Material are the inputs to the `SetMaterial` subroutine.
   `ObjMaterialMngt` object is of type `StrMaterialMngt`. It is retrieved from the stiffener `iObjProductStiffener`. `SetMaterial` method is called to set the
   on object `ObjMaterialMngt`.

   iii. Similar to setting the material, to set the category, we use the `SetCategory` subroutine in the script. This method retrieves the `StrCategoryMngt` object fror
   `StrSddStiffenerOnFreeEdge` object and sets the category using `SetCategory` method on the `StrCategoryMngt` object.

```
...
    ' Get StrSddStiffenerOnFreeEdge object
    Dim ObjStrSddStiffenerOnFreeEdge As StrSddStiffenerOnFreeEdge
    Set ObjStrSddStiffenerOnFreeEdge = oObjProdStiffener.StrSddStiffenerOnFreeEdge
    ' Set Category
    SetCategory ObjStrSddStiffenerOnFreeEdge, "SddStiffenerOnFreeEdge"
...
```

   The `SetCategory` subroutine

```
...
Sub SetCategory(iObjStrSddStiffenerOnFreeEdge As StrSddStiffenerOnFreeEdge, iCategory As String)
    ' Get the StCategoryMngt object
    Dim ObjStrCategoryMngt As StrCategoryMngt
```

```
    Set ObjStrCategoryMngt = iObjStrSddStiffenerOnFreeEdge.StrCategoryMngt
    ' Set category of the stiffener on free edge
    ObjStrCategoryMngt.SetCategory iCategory
End Sub
...
```

A `SddProductStiffenerOnFreeEdge` and a `String` indicating the Category are the inputs to the `SetCategory` subroutine.

iv. Set the `ProfileType` property of the stiffener on free edge to catStrProfileModeOnLimits (here catStrProfileModeOnLimits means profile is created on the plate).

```
...
  ' Set profile type
  SetProfileType ObjStrSddStiffenerOnFreeEdge, catStrProfileModeOnLimits
...
```

The `SetProfileTpye` subroutine.

```
...
Sub SetProfileType(iObjStrSddStiffenerOnFreeEdge As StrSddStiffenerOnFreeEdge, iType As Long)
    ' Set the profile type on the StrSddStiffenerOnFreeEdge
    iObjStrSddStiffenerOnFreeEdge.ProfileType = iType
End Sub
...
```

A `SddProductStiffenerOnFreeEdge` and a `Long` indicating the Profile type are the inputs to the `SetProfileType` subroutine. Pre-defined constants can be
the second parameter.
For e.g. Here we have used `catStrProfileModeOnLimits`

v. Next step is to set the section parameters of the Stiffener On Free Edge. We use the `SetSectionParameters` subroutine to achieve this. The subroutine retri
`StrSectionMngt` object from the created stiffener on free edge object and sets the different section parameters like section name, anchor point, web orientat
flange orientation.

```
...
  ' Set section parameters
  SetSectionParameters ObjStrSddStiffenerOnFreeEdge, "WT18x179.5", "catStrTopCenter"
...
```

The `SetSectionParameters` subroutine

```
...
Sub SetSectionParameters(iObjStrSddStiffenerOnFreeEdge As StrSddStiffenerOnFreeEdge, iStrSecName As String, iAnchorPoint As S
    ' Get StrSectionMngt object
    Dim ObjStrSectionMngt As StrSectionMngt
    Set ObjStrSectionMngt = iObjStrSddStiffenerOnFreeEdge       .StrSectionMngt
    ' Set different section parameters
    ObjStrSectionMngt.SetSectionName  iStrSecName
    ObjStrSectionMngt.AnchorPoint = iAnchorPoint

    ' Set the rest as deault
    ObjStrSectionMngt.WebOrientation = 1
    ObjStrSectionMngt.FlangeOrientation = 1
    ObjStrSectionMngt.WebOrientation = 1
    ObjStrSectionMngt.AngleMode = 0
End Sub
...
```

The `SetSectionParameters` takes a StrSddStiffenerOnFreeEdge, a string indicating the section name and another string indicating the anchor point for the
on free edge as its inputs. The other parameters, as seen, are set to their default values. The `SetSectionParameters` could however, be extended to receive
parameter to make the script more versatile.
Within the subroutine, `ObjStrSectionMngt` object which is of type `StrSectionMngt` is used to set the different section properties of the stiffener. `SetSect`
method sets the section name for the stiffener. `AnchorPoint` property is used to set anchor point to "catStrTopCenter". Similarly `WebOrientation` and
`FlangeOrientation` properties are set to 1.

vi. Further, we set the support plate and select the limit on which we wish to create the stiffener on free edge. We use the `SetSupportPlate` subroutine. It retri
`StrProfileOnLimits` object from the created stiffener on free edge object and sets the support plate and the limit.

```
...
Sub SetSupportPlate(iObjStrSddStiffenerOnFreeEdge As StrSddStiffenerOnFreeEdge, iObjSddPlate1 As SddPlate)
  ' Get the StrProfileOnLimits object
    Dim ObjStrProfileOnLimits As StrProfileOnLimits
    Set ObjStrProfileOnLimits = iObjStrSddStiffenerOnFreeEdge.StrProfileOnLimits

   ' Set the support plate
   Dim ObjStrPlate As StructurePlate
   Set ObjStrPlate = iObjSddPlate1
   ObjStrProfileOnLimits.ReferenceSupportPlate = ObjStrPlate

   ' Set the edge Limit
   Dim LimList(0) As Variant
   LimList(0) = 4
   ObjStrProfileOnLimits.SetLimits (LimList)
End Sub
...
```

`StrProfileOnLimits` is obtained from `iObjStrSddStiffenerOnFreeEdge` and `ObjStrPlate` is obtained from `iObjSddPlate1`.
Now, `ObjStrPlate` is set as the support to `ObjStrProfileOnLimits` using the `ReferenceSupportPlate` property. Further, to set the limit, we create a list
indices and call the `SetLimits` method with the list as the parameter. The Stiffener On Free Edge will be created on every edge mentioned through the list c

vii. By now the Stiffener On Free Edge gets created. However, it is created upto the limits of the plate.
In case we wish to set the limits explicitly, we can obtain the reference planes from the `RfgService` service and apply as limits to stiffener on free edge.
The code below shows how this can be done:

```
...
    ' Get Service manager
    Dim Manager As RfgService
    Set Manager = CATIA.ActiveEditor.GetService("RfgService")
    ' Set the limits
```

```
    Dim StartLimit As Reference
    Dim EndLimit As Reference
    Set ObjStartLimit = Manager.GetReferencePlane(ObjPart, 3, "LONG.14")
    Set StartLimit = ObjPart.CreateReferenceFromObject(ObjStartLimit)
    Set ObjEndLimit = Manager.GetReferencePlane(ObjPart, 3, "LONG.-14")
    Set EndLimit = ObjPart.CreateReferenceFromObject(ObjEndLimit)
    ' Set the limits on stiffener on free edge
    SetLimits ObjStrSddStiffenerOnFreeEdge, StartLimit, EndLimit
...
```

The `SetLimits` subroutine

```
...
Sub SetLimits(iObjStrSddStiffenerOnFreeEdge As StrSddStiffenerOnFreeEdge, iStartLimit As Reference, iEndLimit As Reference )
    ' Get the StrProfileLimitMngt object
    Dim ObjStrProfileLimitMngt As StrProfileLimitMngt
    Set ObjStrProfileLimitMngt = iObjStrSddStiffenerOnFreeEdge.StrProfileLimitMngt

      ' Set the two limits
    ObjStrProfileLimitMngt.SetLimitingObject 1, iStartLimit
    ObjStrProfileLimitMngt.SetLimitingObject 2, iEndLimit
End Sub
...
```

Input to the `SetLimits` subroutine is a StrSddStiffenerOnFreeEdge, a Reference to start limit and a Reference to end limit.
The subroutine obtains the StrProfileLimitMngt object and calls the `SetLimitingObject` method on it to set the two limits as shown above.

viii. Updates the created stiffener on free edge before exiting subroutine CreateStiffenerOnFreeEdge_PlateLimit.

```
...
  oObjProdStiffener.Update
End Sub
...
```

7. **Creates a Stiffener On Free Edge on an opening on the Plate**

Call `CreateStiffenerOnFreeEdge_PlateOpening` method to create stiffener. Inputs to `CreateStiffenerOnFreeEdge_PlateOpening` method are a SddStiffene object and an empty SddProductStiffenerOnFreeEdge object. When the function executes, `ObjSddProductStiffenerOnFreeEdge_PlateOpening` stores the Stiff Free Edge object created on on of the Limits of the Plate.

```
...
  Dim ObjSddProductStiffenerOnFreeEdge_PlateOpening As SddProductStiffenerOnFreeEdge
  CreateStiffenerOnFreeEdge_PlateOpening ObjSddStiffenerMngt, ObjSddProductStiffenerOnFreeEdge_PlateOpening
...
```

The method `CreateStiffenerOnFreeEdge_PlateOpening` is detailed as in the below sub steps.

```
...
  Sub CreateStiffenerOnFreeEdge_PlateOpening(iObjSddStiffenerMngt As SddStiffenerMngt, oObjProdStiffener As SddProductStiffenerOnF
...
```

i. Create a new empty stiffener by the method `AddStiffenerOnFreeEdge` from object `SddStiffenerMngt`

```
...
  Sub CreateStiffenerOnFreeEdge(iObjSddStiffenerMngt As SddStiffenerMngt, oObjProductStiffener As SddProductStiffenerOnFreeEd
  ' Add Stiffener On Free Edge
  Set oObjProductStiffener = iObjSddStiffenerMngt.AddStiffenerOnFreeEdge
...
```

On `iObjSddStiffenerMngt` object `AddStiffenerOnFreeEdge` method is called to create the empty stiffener. Now UC needs to set the different properties stiffener like material, category etc.

ii. Similar to the ObjSddProductStiffenerOnFreeEdge_PlateLimit properties set above, we set the properties for ObjSddProductStiffenerOnFreeEdge_PlateOpe Set the material.

```
...
  ' Set material
  SetMaterial oObjProdStiffener, "Steel A42"
...
```

The `SetMaterial` subroutine has been detailed above under Step 6.

iii. We now move on to setting the category.

```
...
  ' Set category
  SetCategory ObjStrSddStiffenerOnFreeEdge, "SddStiffenerOnFreeEdge"
...
```

The `SetCategory` subroutine has been detailed above under Step 6.

iv. Set the `ProfileType` property of the stiffener on free edge to catStrProfileModeOnOpening (here catStrProfileModeOnOpening means profile is created on opening present on the plate).

```
...
  ' Set profile type
  SetProfileType ObjStrSddStiffenerOnFreeEdge, catStrProfileModeOnOpening
...
```

The `SetProfileTpye` subroutine has been detailed above under Step 6.

A `SddProductStiffenerOnFreeEdge` and a `Long` indicating the Profile type are the inputs to the `SetProfileType` subroutine. Pre-defined constants can be the second parameter.
For e.g. Here we have used `catStrProfileModeOnOpening`

v. Next step is to set the section parameters of the Stiffener On Free Edge. We use the `SetSectionParameters` subroutine to achieve this. The subroutine retri `StrSectionMngt` object from the created stiffener on free edge object and sets the different section parameters like section name, anchor point, web orientat

flange orientation.

```
...
  ' Set section parameters
  SetSectionParameters ObjStrSddStiffenerOnFreeEdge, "WT18x179.5", "catStrTopCenter"
...
```

The `SetSectionParameters` subroutine has been detailed above under Step 6.

The `SetSectionParameters` takes a StrSddStiffenerOnFreeEdge, a string indicating the section name and another string indicating the anchor point for the on free edge as its inputs. The other parameters, as seen, are set to their default values. The `SetSectionParameters` could however, be extended to receive parameter to make the script more versatile.

vi. Further, we set the support opening on which we wish to create the stiffener on free edge. We use the `SetSupportOpening` subroutine. It retrieves the `StrProfileOnOpening` object from the created stiffener on free edge object and sets the opening as the support.

```
...
Sub SetSupportOpening(iObjStrSddStiffenerOnFreeEdge As StrSddStiffenerOnFreeEdge, iObjSddPlate1 As SddPlate)
  ' Get the StrProfileOnOpening object
  Dim ObjStrProfileOnLimits As StrProfileOnOpening
  Set ObjStrProfileOnLimits = iObjStrSddStiffenerOnFreeEdge.StrProfileOnOpening

  ' Get the opening form the plate
  Dim ObjStrOpenings As StrOpenings
  Set ObjStrOpenings = iObjSddPlate1.GetOpenings(0)
  Dim ObjStrOpening As StrOpening
  Set ObjStrOpening = ObjStrOpenings.Item(1)
  ObjStrProfileOnOpening.Opening = ObjStrOpening
...
```

`StrProfileOnOpening` is obtained from `iObjStrSddStiffenerOnFreeEdge` and `ObjStrOpening` us obtained from `GetOpenings` method on `iObjSddPlat` This `ObjStrOpening` is set as the `Opening` property of `ObjStrProfileOnOpening`.

vii. Updates the created stiffener on free edge before exiting subroutine CreateStiffenerOnFreeEdge_PlateOpening.

```
...
  oObjProdStiffener.Update
End Sub
...
```

**Fig.1: Stiffener On Free Edge on a Plate Limit with limits**

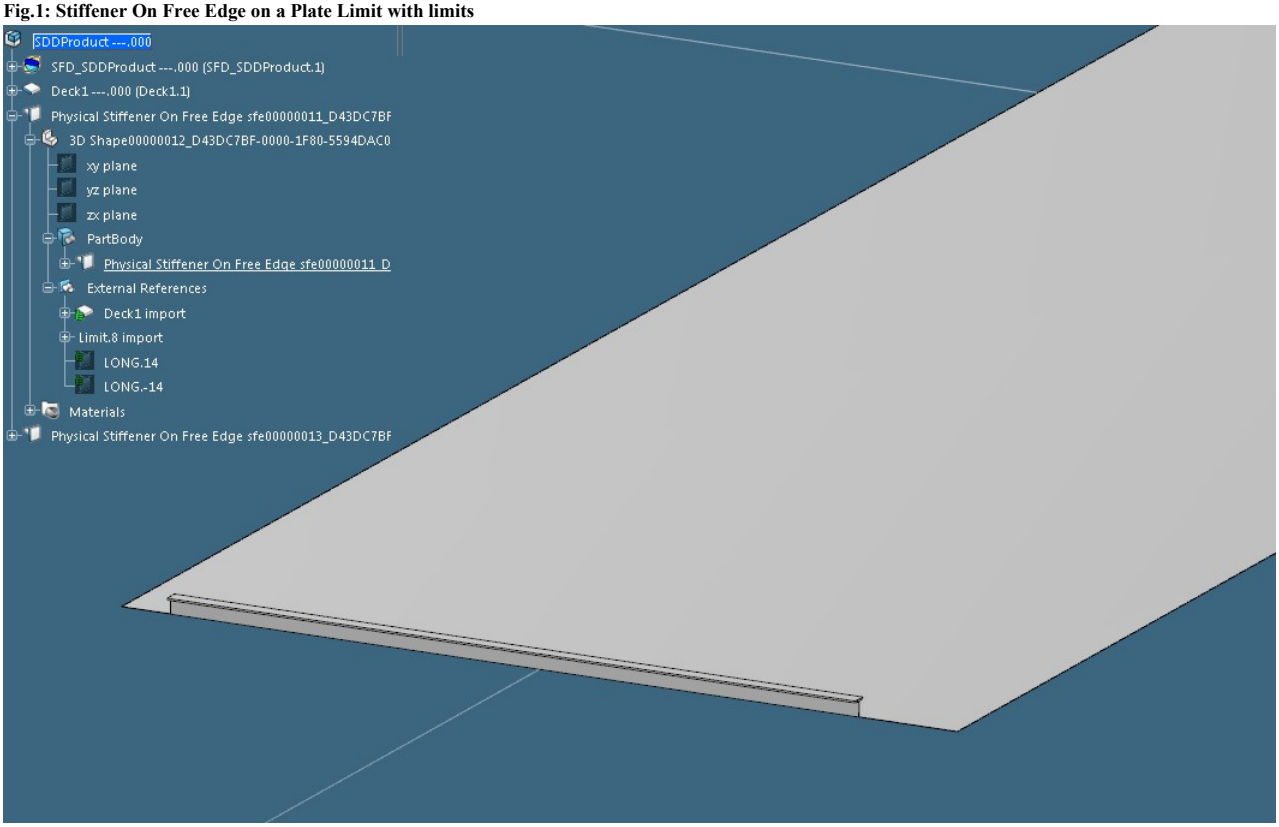
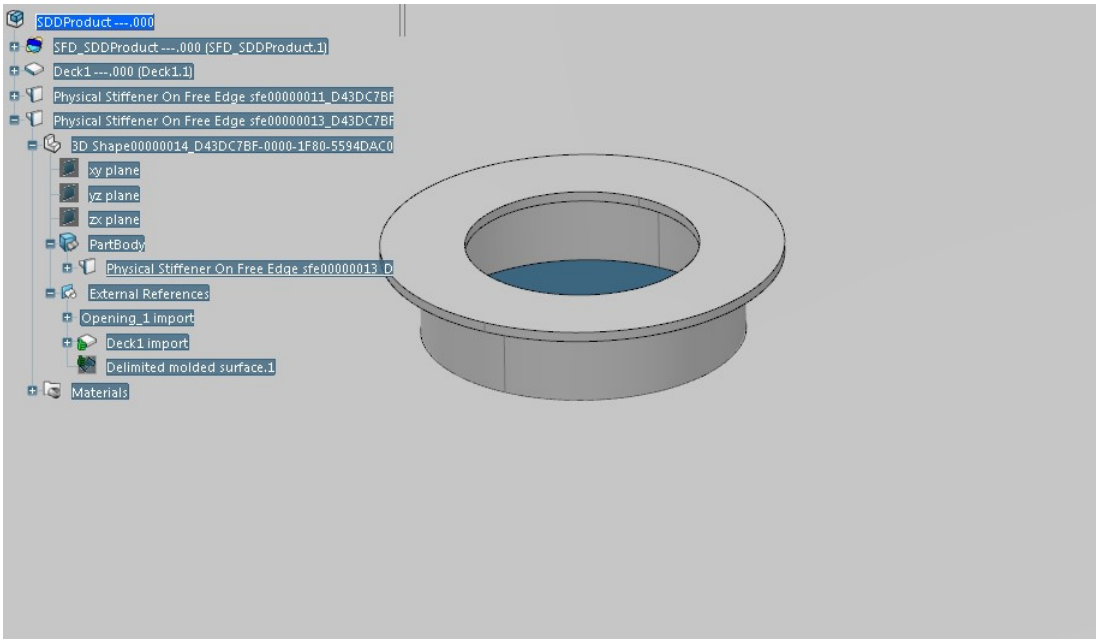



**Fig.2: Stiffener On Free Edge on an opening on the Plate**




# Creating an SDD Flange

This use case primarily focuses on the methodology to create a SDD flange.

Related Topics
Structure Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcCGR.3dxml`, `CAAScdSddUcSR.3dxml`, `CAAScdSddUcSteel_A42.3dxml`, `CAAScdSddUcWT18x179_5.3dxml`, `CAAScdSddUcCreateFlange.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\`
  where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcCreateFlangeSource.htm

This use case can be divided in 6 steps:

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Part object
3. Retrieves Service Manger (RfgService)
4. Retrieves a SddProduct Plate object
5. Create Flange
6. Update the plate

1. **Searches and opens model which is named as "SddProduct"**

   As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim SDDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SDDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SDDPrdEditor`**, a Editor object. After searching and opening of SDD model from underlying database the current active editor is returned in **`SDDPrdEditor`**.

2. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Dim product1Service  As  PLMProductService
  Set product1Service = DDPrdEditor.GetService("PLMProductService")
  Dim ObjVPMRootOccurrence  As  VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference  As  VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances  As  VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

3. **Retrieves Service manager (RfgService)**

   In this step UC retrieves `RfgService`.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

`GetService` method returns `RfgService`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

4. **Retrieves a SddProduct Plate object**

   In this step Uc retrieves the SddProduct Plate object.

```
...
  Set ListOfInstances = ObjVPMReference.Instances
  Set PlateRef = ListOfInstances.Item(1).ReferenceInstanceOf
  Set PlateRepInstances = PlateRef.RepInstances
  Set PlateRepInstReference = PlateRepInstances.Item(1).ReferenceInstanceOf
  Set PlatePart = PlateRepInstReference.GetItem("Part")

  Dim ObjSddProductPlate As SddProductPlate
  SFDProdSel.Add PlateRef
  Set ObjSddProductPlate = SFDProdSel.FindObject("CATIASddProductPlate")
  Dim ObjSddPlate As SddPlate
  Set ObjSddPlate = ObjSddProductPlate.SddPlate
...
```

   In this step Uc retrieves the SddProduct Plate as shown above.

5. **Create Flange**

   Now plate is available to create flange on it. Call `CreateFlange` method to create flange. `CreateFlange` method takes a plate as input and created flange is returned in output parameter.

```
...
  Dim ObjStrFlange As StrFlange
  CreateFlange ObjSddPlate, ObjStrFlange
...
```

   `oObjStrFlange` object is retreived from `AddFlange`.

```
Sub AddFlange(iObjSddPlate As SddPlate, oObjStrFlange As StrFlange)

  'Get StrFlanges object
  Dim ObjStrFlanges As StrFlanges
  Set ObjStrFlanges = iObjSddPlate.Flanges

  'Create a flange
  Set oObjStrFlange = ObjStrFlanges.Add

End Sub
```

   `StrFlanges` object is retrieved in `ObjStrFlanges` variable from `iObjSddPlate` object. On `ObjStrFlanges` object **`Add`** method is called to create the empty flange. Now Uc needs to set the different properties of the flange like Type, WidthMeasurementType, Start Limit, End Limit, etc.

```
...
  'Add Flange
   AddFlange iObjSddPlate, oObjStrFlange

    'Set Type
    oObjStrFlange.Type = 1

    'Set FlangeStartLimit
    Set RefSddPlane = ObjPart.FindObjectByName("CROSS.40")
    Dim ObjSddReferencePlane_1 As Reference
    SFDProdSel.Add RefSddPlane
    Set ObjSddReferencePlane_1 = SFDProdSel.FindObject("CATIAReference")
    oObjStrFlange.FlangeStartLimit = ObjSddReferencePlane_1

    'Set FlangeEndLimit
    Set RefSddPlane = ObjPart.FindObjectByName("CROSS.24")
    Dim ObjSddReferencePlane_2 As Reference
    SFDProdSel.Add RefSddPlane
    Set ObjSddReferencePlane_2 = SFDProdSel.FindObject("CATIAReference")
    oObjStrFlange.FlangeEndLimit = ObjSddReferencePlane_2

    'Set WidthMeasurementType
    oObjStrFlange.WidthMeasurementType = 1

    'Get OperatedPlate
    Dim ObjSddPlate_1 As Reference
    Set ObjSddPlate_1 = oObjStrFlange.OperatedPlate

    'Get Type
    Dim oType As long
    oType = oObjStrFlange.Type

    'Get BendingAngle
    Dim oBendingAngle As Parameter
    Set oBendingAngle = oObjStrFlange.BendingAngle
    oBendingAngle.ValuateFromString("120deg")

    'Get BendingRadius
    Dim oBendingRadius As Parameter
    Set oBendingRadius = oObjStrFlange.BendingRadius
    oBendingRadius.ValuateFromString("8mm")

    'Get FlangeStartLimit
    Dim oFlangeStartLimit As Reference
    Set oFlangeStartLimit = oObjStrFlange.FlangeStartLimit

    'Get FlangeEndLimit
    Dim oFlangeEndLimit As Reference
    Set oFlangeEndLimit = oObjStrFlange.FlangeEndLimit

    'Get FlangeWidth
    Dim oFlangeWidth As Parameter
    Set oFlangeWidth = oObjStrFlange.FlangeWidth
    oFlangeWidth.ValuateFromString("500mm")
```

```
    'Get WidthMeasurementType
    Dim oWidthMeasurementType As long
    Set oWidthMeasurementType = oObjStrFlange.WidthMeasurementType

    'Get StartEndCutOffset
    Dim oStartEndCutOffset As Parameter
    Set oStartEndCutOffset = oObjStrFlange.StartEndCutOffset
    oStartEndCutOffset.ValuateFromString("10mm")

    'Get EndEndCutOffset
    Dim oEndEndCutOffset As Parameter
    Set oEndEndCutOffset = oObjStrFlange.EndEndCutOffset
    oEndEndCutOffset.ValuateFromString("10mm")

    'Get StartEndCutRadius
    Dim oStartEndCutRadius As Parameter
    Set oStartEndCutRadius = oObjStrFlange.StartEndCutRadius
    oStartEndCutRadius.ValuateFromString("50mm")

    'Get EndEndCutRadius
    Dim oEndEndCutRadiusAs Parameter
    Set oEndEndCutRadius= oObjStrFlange.EndEndCutRadius
    oEndEndCutRadius.ValuateFromString("50mm")

    'Get StartEndCutDistance
    Dim oStartEndCutDistance As Parameter
    Set oStartEndCutDistance = oObjStrFlange.StartEndCutDistance
    oStartEndCutDistance.ValuateFromString("100mm")

    'Get EndEndCutDistance
    Dim oEndEndCutDistance As Parameter
    Set oEndEndCutDistance = oObjStrFlange.EndEndCutDistance
    oEndEndCutDistance.ValuateFromString("100mm")

    'Get StartEndCutAngle
    Dim oStartEndCutAngle As Parameter
    Set oStartEndCutAngle = oObjStrFlange.StartEndCutAngle
    oStartEndCutAngle.ValuateFromString("60deg")

    'Get EndEndCutAngle
    Dim oEndEndCutAngle As Parameter
    Set oEndEndCutAngle = oObjStrFlange.EndEndCutAngle
    oEndEndCutAngle.ValuateFromString("60deg")

    'Set Edges
    Dim EdgeList(1) As Variant
    EdgeList(0) = 2
    EdgeList(1) = 3
    oObjStrFlange.SetEdges EdgeList

    'Get Edge
    Dim ObjSddReferencePlane_3 As Reference
    Set ObjSddReferencePlane_3 = oObjStrFlange.Edge
...
```

`Type`: Set and get the Type of conversion for flange. 1- Centered, 2- Tangent

`FlangeStartLimit`: Set and get the Flange Start Limit.

`FlangeEndLimit`: Set and get the Flange End Limit.

`WidthMeasurementType`: Set and get the Width Measurement Type. 1- FlangeWidthToInnerFace, 2- FlangeWidthToOuterFace, 3- FlangeWidthToNeutralFibre

`OperatedPlate`: Get the operated Plate.

`BendingAngle`: Get the Bending Angle and valuate it.

`BendingRadius`: Get the Bending Radius and valuate it.

`FlangeWidth`: Get the Flange Width and valuate it.

`StartEndCutOffset`: Get the Start EndCut Offset and valuate it.

`EndEndCutOffset`: Get the End EndCut Offset and valuate it.

`StartEndCutRadius`: Get the Start EndCut Radius and valuate it.

`EndEndCutRadius`: Get the End EndCut Radius and valuate it.

`StartEndCutDistance`: Get the Start EndCut Distance and valuate it.

`EndEndCutDistance`: Get the End EndCut Distance and valuate it.

`StartEndCutAngle`: Get the Start EndCut Angle and valuate it.

`EndEndCutAngle`: Get the End EndCut Angle and valuate it.

`SetEdges`: Set the Edges on which flanges to be created.

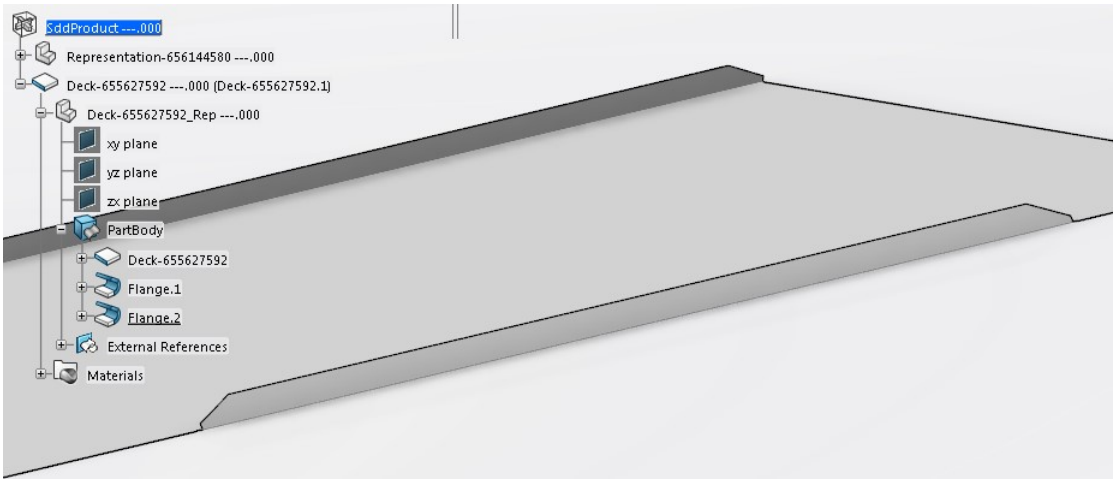
`Edge`: Get the Edge of Plate.

6. **Update the plate**

```
...
    ObjSddProductPlate.Update
End Sub
```

Method `Update` updates the Product Plate.

Fig.1: Created Flanges




# Creating an SDD Bracket

This use case primarily focuses on the methodology to create a SDD bracket.

Related Topics
Structure Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcCreateBracket.3dxml`, `CAAScdSddUcSteel_A90.3dxml`,Bracket_KN6.3dxm`l`, `CAAScdSddUcWT18x179_5.3dxml`, `CAAScdSddUcCGR.3dxml` and `CAAScdSddUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcCreateBracketSource.htm

This use case can be divided in seven steps:

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Part object
3. Retrieves Service Manger (RfgService)
4. Retrieves two Product Stiffener objects
5. Get SddFactory
6. Create Contour Based
7. Update the product bracket

1. **Searches and opens model which is named as "SddProduct"**

   As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim SFDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SFDPrdEditor
```

   The function **`OpenProduct`** returns **`SFDPrdEditor`**, a Editor object. After searching and opening of SDD model from underlying database the current active editor

2. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Set SFDProdSel = SFDPrdEditor.Selection
  Dim product1Service  As  PLMProductService
  Set product1Service = DDPrdEditor.GetService("PLMProductService")
  Dim ObjVPMRootOccurrence  As  VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference  As  VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances  As  VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

3. **Retrieves Service manager (RfgService)**

   In this step UC retrieves `RfgService`.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

   `GetService` method returns `RfgService`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

4. **Retrieves two Product Stiffener objects**

   In this step Uc retrieves the Sdd Product Stiffener objects.

```
...
```

```
  'Get Product Stiffener-1(Stiffener-476667114)
  Set ListOfInstances = ObjVPMReference.Instances
  Set StiffenerRef_1 = ListOfInstances.Item(3).ReferenceInstanceOf
  Set StiffenerRepInstances = StiffenerRef_1.RepInstances
  Set StiffenerRepInstReference = StiffenerRepInstances.Item(1).ReferenceInstanceOf
  Set StiffenerPart_1 = StiffenerRepInstReference.GetItem("Part")
  'get Sdd product stiffener
  Dim ObjSddProductStiffener_1 As SddProductStiffener
  SFDProdSel.Add StiffenerRef_1
  Set ObjSddProductStiffener_1 = SFDProdSel.FindObject("CATIASddProductStiffener")
  Dim ObjSddStiffener_1 As SddStiffener
  Set ObjSddStiffener_1 = ObjSddProductStiffener_1.SddStiffener
  'get reference of Stiffener
  Dim RefObjSddStiffener_1 As Reference
  Set RefObjSddStiffener_1 = StiffenerPart_1.CreateReferenceFromObject(ObjSddStiffener_1)

  'Get Product Stiffener-2(Stiffener-476667115)
  Set ListOfInstances = ObjVPMReference.Instances
  Set StiffenerRef_2 = ListOfInstances.Item(4).ReferenceInstanceOf
  Set StiffenerRepInstances = StiffenerRef_2.RepInstances
  Set StiffenerRepInstReference = StiffenerRepInstances.Item(1).ReferenceInstanceOf
  Set StiffenerPart_2 = StiffenerRepInstReference.GetItem("Part")
  'get Sdd product stiffener
  Dim ObjSddProductStiffener_2 As SddProductStiffener
  SFDProdSel.Add StiffenerRef_2
  Set ObjSddProductStiffener_2 = SFDProdSel.FindObject("CATIASddProductStiffener")
  Dim ObjSddStiffener_2 As SddStiffener
  Set ObjSddStiffener_2 = ObjSddProductStiffener_2.SddStiffener
  'get reference of Stiffener
  Dim RefObjSddStiffener_2 As Reference
  Set RefObjSddStiffener_2 = StiffenerPart_2.CreateReferenceFromObject(ObjSddStiffener_2)
...
```

In this step Uc retrieves the SddProduct Stiffeners as shown above.

5. **Get SddFactory**

   In this step UC retrieves `SddFactory` object.

```
...
  'Get SddFactory for creating SDD objects
  Dim ObjSddFactory As SddFactory
  'Set ObjSddFactory = ObjVPMRootOccurrence.GetItem("SddFactory")
  SFDProdSel.Add ObjVPMRootOccurrence
  Set ObjSddFactory = SFDProdSel.FindObject("CATIASddFactory")
...
```

6. **Create Contour Based**

   Call `CreateContourBased` method to create contour based plate.

```
...
  Dim ObjSddProdBracket As SddProductBracket
  CreateContourBased ObjSddFactory, ObjSddProdBracket, RefObjSddStiffener_1, RefObjSddStiffener_2
...
```

`oObjProdBracket` object is retreived from `AddBracket`.

```
Sub AddBracket(iObjSddFactory As SddFactory, oObjProductbracket As SddProductBracket)
  Set oObjProductBracket = iObjSddFactory.AddProductBracket(5)
End Sub
```

`SddProductBracket` object is retrieved in `oObjProductBracket` variable from `iObjSddFactory` object. On `ObjProdBracket` object **`SddContourBased`** method i
Based Plate. Now Uc needs to set the different properties of the parametric plate like Material, Material Extrusion, Support, Category, Sketch, Thickness and Limi

```
...
Sub CreateContourBased(iObjSddFactory As SddFactory, oObjProdBracket As SddProductBracket, iRefObjSddStiffener_1 As Reference, iRe

  'Call subroutine AddBracket to add a contour based using the SDD Factory
  AddBracket iObjSddFactory, oObjProdBracket
  Dim ObjSddContourBased As SddContourBased
  Set ObjSddContourBased = oObjProdBracket.SddContourBased

  'Set material
  SetMaterial oObjProdBracket, "Steel A42"

  'Set material orientation
  Dim ExtrusionManager As StrPlateExtrusionMngt
  Set ExtrusionManager = ObjSddContourBased.StrPlateExtrusionMngt
  ExtrusionManager.ThrowOrientation = 1

  'Set category
  SetCategory ObjSddContourBased, "Panel"

  'Browse to "LONG.9" reference plane and set it as the support
  Dim PlateSupport As Reference
  Set PlateSupport = Manager.GetReferencePlane(ObjPart, 3, "LONG.0")
  SetContourBasedPlateSupport ObjSddContourBased, PlateSupport

  'Set the DMS
  Dim PlateDMS As StrSketchBasedDMSMngt
  Dim RefSddContourBased As Reference
  Set RefSddContourBased = ObjSddContourBased.Reference
  Set PlateDMS = ObjSddContourBased.StrSketchBasedDMSMngt
  PlateDMS.SetStrSketch2 "SAMPLE_RCO_2LIMITS_CURVED", RefSddContourBased

  'Set thickness
  SetPlateGeometryMngtData ObjSddContourBased, "10mm", 0

  'Set Plate limits
  SetContourBasedPlateLimits ObjSddContourBased, iRefObjSddStiffener_1, iRefObjSddStiffener_2

  'Reverse the ThrowOrientation.
```

```
  Dim ObjSfdPlateGeometryMngt As StrPlateExtrusionMngt
  Set ObjSfdPlateGeometryMngt = ObjSddContourBased.StrPlateExtrusionMngt
  ObjSfdPlateGeometryMngt.ReverseThrowOrientation

End Sub
...
```

Above used methods are elaborated below.

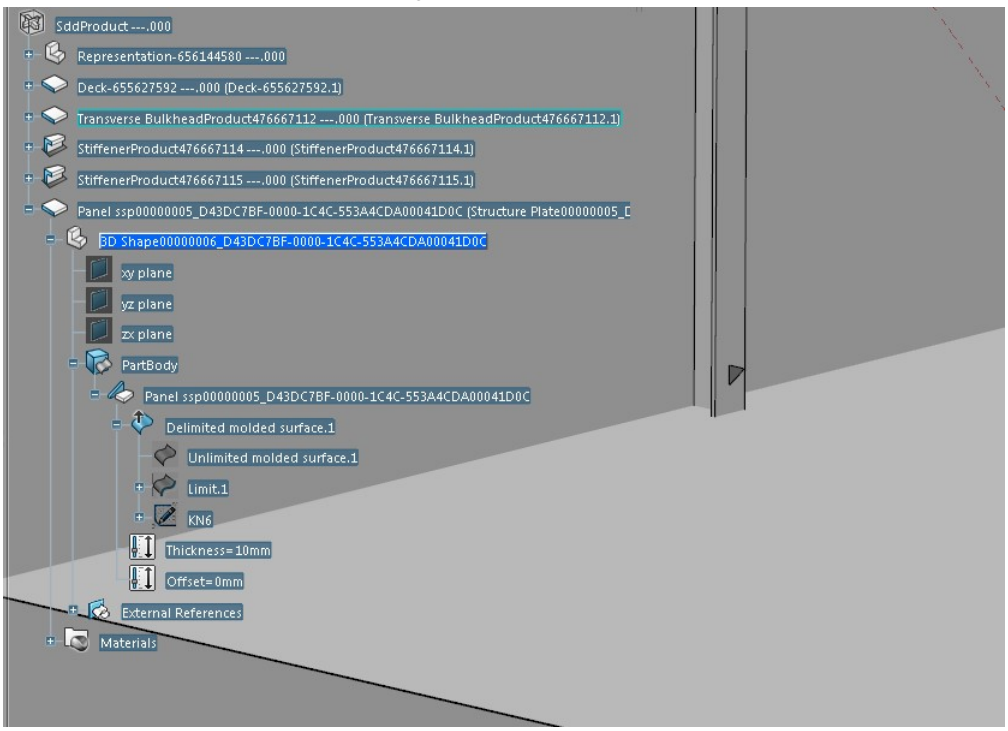
```
...
Sub SetMaterial(iObjSddProdBracket As SddProductBracket, iStrMaterial As String)
  Dim ObjMaterialMngt As StrMaterialMngt
  Set ObjMaterialMngt = iObjSddProdBracket.StrMaterialMngt
  ObjMaterialMngt.SetMaterial (iStrMaterial)
End Sub
...

...
Sub SetCategory(iObjSddContourBased As SddContourBased, iCategory As String)
  Dim ObjStrCategoryMngt As StrCategoryMngt
  Set ObjStrCategoryMngt = iObjSddContourBased.StrCategoryMngt
  ObjStrCategoryMngt.SetCategory iCategory
End Sub
...
Sub SetContourBasedPlateSupport(iObjSddContourBased As SddContourBased, iSupport As Reference)
  Dim ObjSfdPlateSupport As StrPanelSurf
  Set ObjSfdPlateSupport = iObjSddContourBased.StrPanelSurf
  ObjSfdPlateSupport.Support = iSupport
End Sub
...

...
Sub SetPlateGeometryMngtData(iObjSddContourBased As SddContourBased, iThickness As String, iThrowOrientation As Long)
  Dim ObjSfdPlateGeometryMngt As StrPlateExtrusionMngt
  Set ObjSfdPlateGeometryMngt = iObjSddContourBased.StrPlateExtrusionMngt
  ObjSfdPlateGeometryMngt.ThrowOrientation = iThrowOrientation
  Dim Thickness As Parameter
  Set Thickness = ObjSfdPlateGeometryMngt.GetThickness
  Thickness.ValuateFromString iThickness
End Sub
...

...
Sub SetContourBasedPlateLimits(iObjSddContourBased As SddContourBased, iRefObjSddStiffener_1 As Reference, iRefObjSddStiffener_2 A
  'Obtain the StrPanelLimitMngt
  Dim ObjSfdPlateLimitMngt As StrPanelLimitMngt
  Set ObjSfdPlateLimitMngt = iObjSddContourBased.StrPanelLimitMngt

  'Apply the References to the 2 SDD stiffeners as limits to the Contour Based
  ObjSfdPlateLimitMngt.SetLimitingObject2 iRefObjSddStiffener_1, 2, 0, -1, "22"
  ObjSfdPlateLimitMngt.SetLimitingObject2 iRefObjSddStiffener_2, 3, 0, -1, "22"
End Sub
...
```

7. **Update the product bracket**

```
...
  ObjSddProdBracket.Update
End Sub
```

Method `Update` updates the Product Bracket.

Fig.1: Stiffener with Limits



# Creating Standard Opening On Profile

This use case primarily focuses on the methodology to create Standard Opening On Profile.

Related Topics
Structure Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcCreateOpeningStandard.3dxml`, `CAAScdSddUcCGR.3dxml` and `CAAScdSddUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcCreateOpeningOnProfileStandardSource.htm

This use case can be divided in six steps:

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves service of type RfgService
5. Retrieves a SDD stiffener object
6. Create standard Opening on profile using Offset/Offset positioning strategy

1. **Searches and opens model which is named as "SddProduct"**

   As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim SDDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SDDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SDDPrdEditor`**, a Editor object. After searching and opening of SDD model from underlying database the current active editor

2. **Retrieves Selection object**

   In this step UC retrieves Selection object in `SDDProdSel` variable.

```
...
  Set SDDProdSel = SDDPrdEditor.Selection
...
```

3. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Dim product1Service  As  PLMProductService
  Set product1Service = DDPrdEditor.GetService("PLMProductService")
  Dim ObjVPMRootOccurrence  As  VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference  As  VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances  As  VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

4. **Retrieves service of type RfgService**

   In this step UC retrieves `RfgService`.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

   `GetService` method returns `RfgService`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

5. **Retrieves a SDD Stiffener object**

   In this step UC finds a SDD stiffener in the part.

```
...
  Set ListOfInstances = ObjVPMReference.Instances
  Set StiffenerRef = ListOfInstances.Item(2).ReferenceInstanceOf
  Set StiffenerRepInstances = StiffenerRef.RepInstances
  Set StiffenerRepInstReference = StiffenerRepInstances.Item(1).ReferenceInstanceOf
  Set StiffenerPart = StiffenerRepInstReference.GetItem("Part")
' get Sdd Stiffener
  Dim ObjSddProductStiffener As SddProductStiffener
  SDDProdSel.Add StiffenerRef
  Set ObjSddProductStiffener = SDDProdSel.FindObject("CATIASddProductStiffener")
  Dim ObjSddStiffener As SddStiffener
  Set ObjSddStiffener = ObjSddProductStiffener.SddStiffener
' Get reference of stiffener
  Dim RefObjSddStiffener As Reference
  Set RefObjSddStiffener = StiffenerPart.CreateReferenceFromObject(ObjSddStiffener)
...
```

   In above lines, `FindObject` method finds object whose type is "CATIASddProductStiffener" and returns to it. To retrieve `SddStiffener` object from it, call `SddS` This will give the `SddStiffener` object.

6. **Create Standard Opening On Profile with Offset/Offset positioning strategy**

Call `CreateStandardOpeningOnProfileOffsetOffset` method to create a standard opening on profile with offset/offset positioning strategy. `CreateStandardO` method takes a panel object as input parameter and returns created opening as output parameter in `ObjStrOpeningOnProfile`.

```
...
  Dim ObjStrOpeningOnProfileOffsetOffset As StrOpeningOnProfile
  CreateStandardOpeningOnProfileOffsetOffset ObjSddStiffener, RefObjSddStiffener, ObjStrOpeningOnProfileOffsetOffset
  ObjSddProductStiffener.Update
...
```

The method `CreateStandardOpeningOnProfileOffsetOffset` is detailed as in the below sub steps.

```
...
Sub CreateStandardOpeningOnProfileOffsetOffset(iObjSddStiffener As SddStiffener, iRefObjSddStiffener As Reference, oObjStrOpeningO

    'Add Opening On Profile
    AddOpening iObjSddStiffener, oObjStrOpeningOnProfile

    'Get StrOpening from StrOpeningOnProfile
    Dim ObjStrOpening As StrOpening
    Set ObjStrOpening = oObjStrOpeningOnProfile.StrOpening

    'set opening type catStrOpeningModeStandard for standard opening
    ObjStrOpening.OpeningType = catStrOpeningModeStandard

    'set category
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = ObjStrOpening.StrCategoryMngt
    ObjStrCategoryMngt.SetCategory "ManHole"

    'Get StrOpeningStandard from StrOpening
    Dim ObjStrOpeningStandard As StrOpeningStandard
    Set ObjStrOpeningStandard = ObjStrOpening.StrOpeningStandard

    'set standard mode type
    ObjStrOpeningStandard.StandardModeType = catStrOpeningSTDRectMode

    'Get contour and set parameters data
    Dim ContourName As String
    Dim StdContourParms As StrStandardContourParameters
    'get contour names
    Dim ContourNames() As Variant
    Dim ObjStrOpeningsMgr As StrOpeningsMgr
    Set ObjStrOpeningsMgr = iObjSddStiffener.StrOpeningsMgr
    'get available contour names
    ObjStrOpeningsMgr.GetAvailableStandardContours ContourNames
    ContourName = ContourNames(1)
   'get parameters of contour
    Set StdContourParms = ObjStrOpeningsMgr.GetStandardContourParms(ContourName)
    'set contour parameters data
    SetRectContourParamsData StdContourParms, "100mm", "50mm", "5mm"

    'Get positioning strategy and set parameters data
    Dim PosStratName As String
    Dim PosStratParms As StrStandardPosStrategyParameters
    'get std positiong strategy names
    Dim StdPosStrategyNames() As Variant
    'get available standard positioning strategy names
    ObjStrOpeningsMgr.GetAvailableStandardPositioningStrategiesForProfile StdPosStrategyNames
    'select positiong strategy offset offset
    PosStratName = StdPosStrategyNames(2)
    'get standard positioining strategy parameters
    Set PosStratParms = ObjStrOpeningsMgr.GetStandardPositioningStrategyParmsForProfile(PosStratName)

    'set standard opening remaining parameters
    'set direction
    ObjStrOpeningStandard.DirectionForOpeningOnProfile = FALSE
    'set extrusion mode
    'Set ObjStrOpeningExtrusionMngt = ObjStrOpening.StrOpeningExtrusionMngt
    'ObjStrOpeningExtrusionMngt.ExtrusionMode = 0
    'set limit mode
    ObjStrOpeningStandard.LimitMode = 0
    'set contour and position strategy for profile
    ObjStrOpeningStandard.SetContourAndPosStrategyForProfile ContourName, StdContourParms, PosStratName, PosStratParms

    'set Offset Offset Position Strategy Parameters
    Set ObjRefElem = Manager.GetReferencePlane(ObjPart, 3, "LONG.4")
    Set RefElem = ObjPart.CreateReferenceFromObject(ObjRefElem)
    ObjStrOpeningStandard.SetOffsetOffsetPosStratParms iRefObjSddStiffener, RefElem, "0mm", "0mm", "Gravity", "0deg"

    'Update created opening on profile object
    ObjPart.UpdateObject oObjStrOpeningOnProfile

End Sub
...
```

`DirectionForOpeningOnProfile`: Sets the direction for opening on profile.

`StandardModeType`: Sets the mode for standard opening on profile. 0 -> catStrOpeningSTDUndefinedMode, 1 -> catStrOpeningSTDRoundMode, 2 -> catStrOpe catStrOpeningSTDOblongMode, 4 -> catStrOpeningSTDCatalogMode

`SetContourAndPosStrategyForProfile`: Sets the contour and position strategy for opening on profile.

`SetOffsetOffsetPosStratParms`: There are in all 6 parameters for this method. 1. The reference of the profile, 2.The reference element used, 3. The horizontal 4. The vertical offset distance from anchor point, 5. The anchor point type, 6. The axis angle.

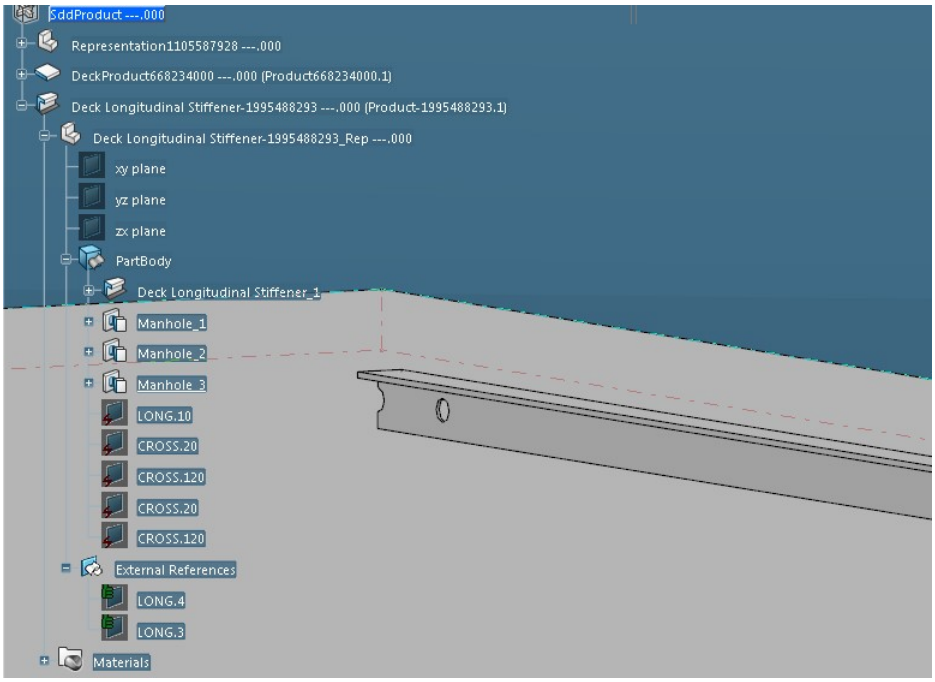
Simillarly, for **Mid Dist/Offset** and **Spacing/Offset** Position Strategy cases `SetMidDistOffsetPosStratParms` and `SetSpacingOffsetPosStratParms` methods

`SetMidDistOffsetPosStratParms`: There are in all 6 parameters for this method. 1. The reference of the profile, 2.The first reference element used, 3. The secon vertical offset distance from anchor point, 5. The anchor point type, 6. The axis angle.

`SetSpacingOffsetPosStratParms`: There are in all 7 parameters for this method. 1. The reference of the profile, 2. Whether reference point is from start or end, Whether mode is absolute or relative, 5. The vertical offset distance from anchor point, 6. The anchor point type, 7. The axis angle.

7. The source also includes standard opening on profile creation using positioning strategies Mid Dist/Offset, Spacing/Offset. To see the source go to CAAScdSddUcCreateOpeningOnProfileStandardSource.htm.

Fig.1: Opening on Profile




# Creating an SDD Opening On Profile Using a Sketch Profile

This use case primarily focuses on the methodology to create a SDD Opening On Profile using sketch profile.

Related Topics
Structure Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcCreateOpeningOnProfile.3dxml`, `CAAScdSddUcCGR.3dxml` and `CAAScdSddUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcCreateOpeningOnProfileSketchSource.htm

This use case can be divided in nine steps:

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves Services Manager
5. Retrieves Product Stiffener
6. Retrieves Sdd Stiffener
7. Retrieves profile for creating opening on profile
8. Create Opening On Profile with UpToLast Limit Mode
9. Create Opening On Profile with Dimensions Limit Mode

1. **Searches and opens model which is named as "SddProduct"**

   As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim SDDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SDDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SDDPrdEditor`**, a Editor object. After searching and opening of SDD model from underlying database the current active editor

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SDDProdSel variable. To retrieve the Selection object **`SDDPrdEditor`** is used.

```
 ...
  Set SFDProdSel = SDDPrdEditor.Selection
 ...
```

3. **Retrieves Part object**

   In this step UC retrieves part object.

```
...
  Dim product1Service As PLMProductService
  Set product1Service = SDDPrdEditor.GetService("PLMProductService")
```

```
  Dim ObjVPMRootOccurrence As VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference As VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances As VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

4. **Retrieves Service manager**

   In this step UC retrieves Service manager variable.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

   `GetService` method returns `RfgService`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

5. **Retrieves Product Stiffener**

   In this step UC retrieves a Product Stiffener.

```
...
  Set ListOfInstances = ObjVPMReference.Instances
  Set StiffenerRef = ListOfInstances.Item(2).ReferenceInstanceOf
  Set StiffenerRepInstances = StiffenerRef.RepInstances
  Set StiffenerRepInstReference = StiffenerRepInstances.Item(1).ReferenceInstanceOf
  Set StiffenerPart = StiffenerRepInstReference.GetItem("Part")
...
```

6. **Retrieves Sdd Stiffener**

   In this step UC retrieves a Sdd Stiffener.

```
...
  Dim ObjSddProductStiffener As SddProductStiffener
  SFDProdSel.Add StiffenerRef
  Set ObjSddProductStiffener = SFDProdSel.FindObject("CATIASddProductStiffener")
  Dim ObjSddStiffener As SddStiffener
  Set ObjSddStiffener = ObjSddProductStiffener.SddStiffener
...
```

7. **Retrieves profile for creating opening on profile**

```
...
  Set ProfileSketchOpeningOnProfileUpToLast = StiffenerPart.FindObjectByName("Profile.1")
  Dim RefProfileSketchOpeningOnProfileUpToLast As Reference
  Set RefProfileSketchOpeningOnProfileUpToLast = StiffenerPart.CreateReferenceFromObject(ProfileSketchOpeningOnProfileUpToLast)
...
```

   `FindObjectByName` method finds object whose name is "Profile.1" and returns reference to it. Here `RefProfileSketchOpeningOnProfileUpToLast` is of type `R`
   `RefProfileSketchOpeningOnProfileUpToLast` object from `ProfileSketchOpeningOnProfileUpToLast` object `CreateReferenceFromObject` method is use

8. **Create Opening On Profile with UpToLast Limit Mode**

   Call `CreateOpeningOnProfileSketchUpToLast` method to create opening on stiffener using sketch profile. `CreateOpeningOnProfileSketchDimensions` meth
   sketch profile as input parameters and it returns created opening as output parameter in `ObjStrOpeningOnProfileWtLtMdUpToLast`.

```
...
  Dim ObjStrOpeningOnProfileWtLtMdUpToLast As StrOpeningOnProfile
  CreateOpeningOnProfileSketchUpToLast ObjSddStiffener, RefProfileSketchOpeningOnProfileUpToLast, ObjStrOpeningOnProfileWtLtMdUpTo
  ObjSddProductStiffener.Update
...
```

```
...
  Sub CreateOpeningOnProfileSketchUpToLast(iObjSddStiffener As SddStiffener, RefProfileSketch As Reference, oObjStrOpeningOnProfil

    'Add Opening on profile
    AddOpening iObjSddStiffener, oObjStrOpeningOnProfile

    'Get StrOpening from StrOpeningOnProfile
    Dim ObjStrOpening As StrOpening
    Set ObjStrOpening = oObjStrOpeningOnProfile.StrOpening

    'Set opening type catStrOpeningModeOutputProfile for standard opening
    ObjStrOpening.OpeningType = catStrOpeningModeOutputProfile

    'Set category
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = ObjStrOpening.StrCategoryMngt
    ObjStrCategoryMngt.SetCategory "ManHole"

    'Set forming mode
    'ObjStrOpening.FormingMode = 0
    Set ObjStrOpeningExtrusionMngt = ObjStrOpening.StrOpeningExtrusionMngt
    'catStrOpeningExtrusionNormal = 1
    ObjStrOpeningExtrusionMngt.ExtrusionMode = 1

    'Set output profile
    'Set LimitMode as 0 for UpTOLast
    'Get object of StrOpeningOutputProfile
    Dim ObjStrOpeningOutputProfile As StrOpeningOutputProfile
    Set ObjStrOpeningOutputProfile = ObjStrOpening.StrOpeningOutputProfile
    'Set OutputProfile for sketch opening
    ObjStrOpeningOutputProfile.OutputProfile = RefProfileSketch

    'Set direction
    Set ObjRefDirection = Manager.GetReferencePlane(ObjPart, 2, "CROSS.20")
    Set RefDirection = ObjPart.CreateReferenceFromObject(ObjRefDirection)
```

```
    ObjStrOpeningOutputProfile.Direction = RefDirection

    'Set LimitMode to UpToLast
    ObjStrOpeningOutputProfile.LimitMode = 0

    ObjPart.UpdateObject oObjStrOpeningOnProfile

  End Sub
...
```

Set various required parameters to get the desired opening on profile with UpToLast limit mode.

9. **Create Opening On Profile with Dimensions Limit Mode**

```
...
  Dim ObjStrOpeningOnProfileWtLtMdDim As StrOpeningOnProfile
  CreateOpeningOnProfileSketchDimensions ObjSddStiffener, RefProfileSketchOpeningOnProfileWtLtMdDim, ObjStrOpeningOnProfileWtLtMdD
  ObjSddProductStiffener.Update
...

...
  Sub CreateOpeningOnProfileSketchDimensions(iObjSddStiffener As SddStiffener, RefProfileSketch As Reference, oObjStrOpeningOnProf

    'Add Opening on profile
    AddOpening iObjSddStiffener, oObjStrOpeningOnProfile

    'Get StrOpening from StrOpeningOnProfile
    Dim ObjStrOpening As StrOpening
    Set ObjStrOpening = oObjStrOpeningOnProfile.StrOpening

    'Set opening type catStrOpeningModeOutputProfile for standard opening
    ObjStrOpening.OpeningType = catStrOpeningModeOutputProfile

    'Set category
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = ObjStrOpening.StrCategoryMngt
    ObjStrCategoryMngt.SetCategory "ManHole"

    'Set forming mode
    'ObjStrOpening.FormingMode = 0
    Set ObjStrOpeningExtrusionMngt = ObjStrOpening.StrOpeningExtrusionMngt
    'catStrOpeningExtrusionNormal = 1
    ObjStrOpeningExtrusionMngt.ExtrusionMode = 1

    'Set output profile
    'Set LimitMode to 1 for Dimensions
    'Get object of StrOpeningOutputProfile
    Dim ObjStrOpeningOutputProfile As StrOpeningOutputProfile
    Set ObjStrOpeningOutputProfile = ObjStrOpening.StrOpeningOutputProfile
    'Set OutputProfile for sketch opening
    ObjStrOpeningOutputProfile.OutputProfile = RefProfileSketch

    'Set direction
    Set ObjRefDirection = Manager.GetReferencePlane(ObjPart, 2, "CROSS.20")
    Set RefDirection = ObjPart.CreateReferenceFromObject(ObjRefDirection)
    ObjStrOpeningOutputProfile.Direction = RefDirection

    'Set LimitMode to Dimensions
    ObjStrOpeningOutputProfile.LimitMode = 1

    'Get dimensionsMngt object to set offsets
    Dim ObjStrOpeningLimitDimensionsMngt As StrOpeningLimitDimensionsMngt
    Set ObjStrOpeningLimitDimensionsMngt = ObjStrOpening.StrOpeningLimitDimensionsMngt

    'Set FirstOffset of opening output profile
    Dim FirstOffset As Parameter
    Set FirstOffset = ObjStrOpeningLimitDimensionsMngt.GetFirstOffset
    FirstOffset.ValuateFromString ("1000mm")
    'Set second Offset of opening output profile
    Dim SecondOffsetOffset As Parameter
    Set SecondOffsetOffset = ObjStrOpeningLimitDimensionsMngt.GetSecondOffset
    SecondOffsetOffset.ValuateFromString ("-2000mm")

    'Invert the first offset and second offset
    ObjStrOpeningLimitDimensionsMngt.Invert

    ObjPart.UpdateObject oObjStrOpeningOnProfile

  End Sub
...
```

Set various required parameters to get the desired opening on profile with Dimensions limit mode.

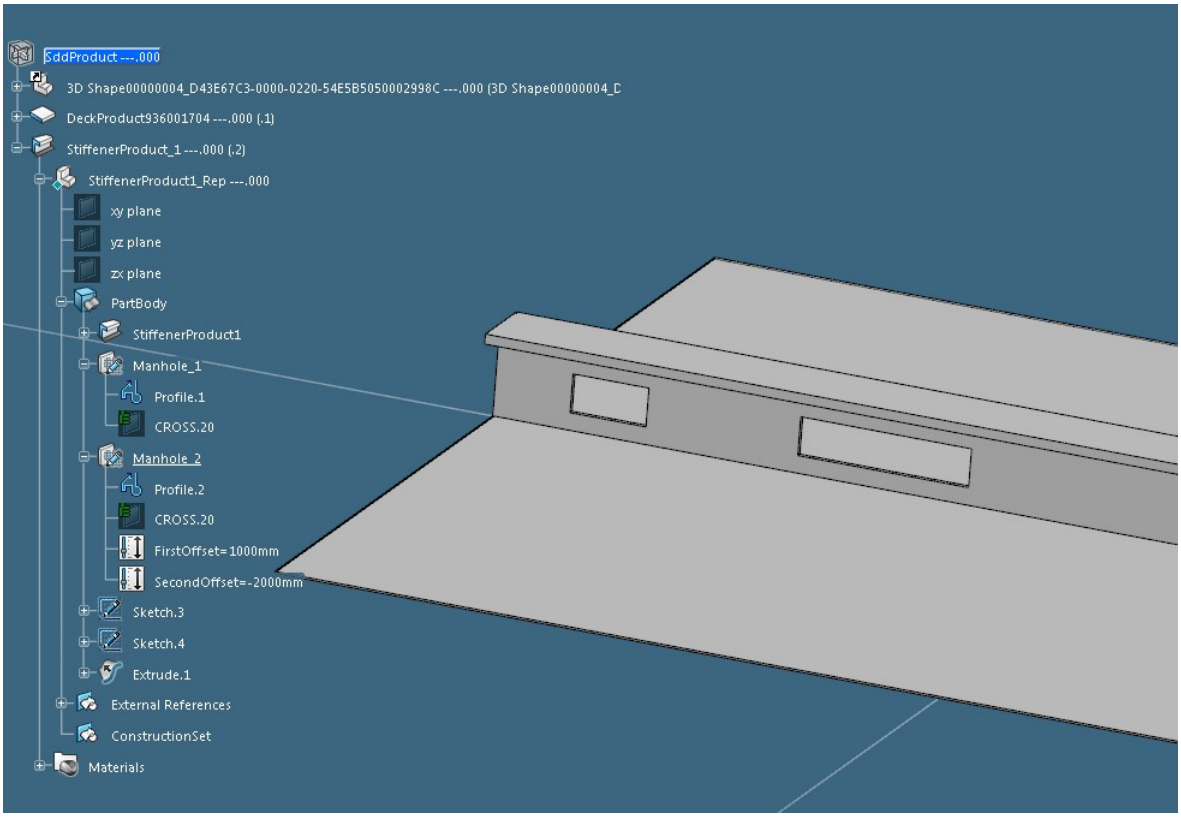
**Detailed steps of method called in the use case**

- **AddOpening method**

```
  Sub AddOpening(iObjSddStiffener As SddStiffener, oObjStrOpeningOnProfile As StrOpeningOnProfile)
    'Get StrOpeningsOnProfile object
    Dim ObjStrOpeningsOnProfile As StrOpeningsOnProfile
    Set ObjStrOpeningsOnProfile = iObjSddStiffener.GetOpeningsOnProfile(0)
    'Add opening
    Set oObjStrOpeningOnProfile = ObjStrOpeningsOnProfile.Add
  End Sub
```

Method `AddOpening`takes a stiffener object `iObjSddStiffener` as input parameter and it returns created opening on profile as output parameter in `oObjStrOpeningOnProfile`. In this method object of type `StrOpeningsOnProfile` is retrieved in `ObjStrOpeningsOnProfile`. Then on this object `Add` method is called to create an opening with no properties set.

Fig.1: Opening using sketch




# Creating an SDD Opening On Profile using 3D Objects

This use case primarily focuses on the methodology to create a SDD Opening on profile using 3D object.

Related Topics
Structure Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcCreateOpeningOnProfile.3dxml`, `CAAScdSddUcCGR.3dxml` and `CAAScdSddUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcCreateOpeningOnProfile3DObjectSource.htm

This use case can be divided in seven steps:

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Selection object
3. Retrieves Part object
4. Retrieves Service manager
5. Retrieves a Stiffener object
6. Get 3D Object for creating opening on profile
7. Creates opening on profile
8. Removes opening on profile
9. Updates the Product Stiffener object

1. **Searches and opens model which is named as "SddProduct"**

   As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim SDDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SDDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SDDPrdEditor`**, a Editor object. After searching and opening of SDD model from underlying database the current active editor **`SDDPrdEditor`**.

2. **Retrieves Selection Object**

   As a next step, the UC retrieves Selection object in SFDProdSel variable. To retrieve the Selection object **`SDDPrdEditor`** is used.

```
 ...
    Set SDDProdSel = SDDPrdEditor.Selection
 ...
```

3. **Retrieves the part object**

   In this step UC retrieves part object.

```
...
```

```
    Dim product1Service As PLMProductService
    Set product1Service = SDDPrdEditor.GetService("PLMProductService")
    Dim ObjVPMRootOccurrence As VPMRootOccurrence
    Set ObjVPMRootOccurrence = product1Service.RootOccurrence
    Dim ObjVPMReference As VPMReference
    Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
    Dim ObjVPMRepInstances As VPMRepInstances
    Set ObjVPMRepInstances = ObjVPMReference.RepInstances
    Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
    Set ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

4. **Retrieves Service manager**

   In this step UC retrieves Service manager variable.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

5. **Retrieves a Stiffener object**

   In this step UC retrieves stiffener object.

```
...
  Set ListOfInstances = ObjVPMReference.Instances
  Set StiffenerRef = ListOfInstances.Item(2).ReferenceInstanceOf
  Set StiffenerRepInstances = StiffenerRef.RepInstances
  Set StiffenerRepInstReference = StiffenerRepInstances.Item(1).ReferenceInstanceOf
  Set StiffenerPart = StiffenerRepInstReference.GetItem("Part")
  'get Sdd Stiffener
  Dim ObjSddProductStiffener As SddProductStiffener
  SFDProdSel.Add StiffenerRef
  Set ObjSddProductStiffener = SFDProdSel.FindObject("CATIASddProductStiffener")
  Dim ObjSddStiffener As SddStiffener
  Set ObjSddStiffener = ObjSddProductStiffener.SddStiffener
...
```

`FindObject` method finds object whose type is "SddProductStiffener" and returns it. To retrieve `SddStiffener` object from `ObjSddProductStiffener` call `SddS` property as shown above. This will return the `SddStiffener` object variable.

6. **Get 3D Object for creating opening on profile**

```
...
    Set Obj3DOpeningOnProfileDimensions = StiffenerPart.FindObjectByName("Extrude.1")
    Dim Ref3DObjectOpeningOnProfileDimensions As Reference
    Set Ref3DObjectOpeningOnProfileDimensions = StiffenerPart.CreateReferenceFromObject(Obj3DOpeningOnProfileDimensions)
...
```

7. **Creates opening on profile**

   Now, stiffener is available to create opening on it. Call `CreateOpeningOnProfile3DObject` method to create opening on profile. `CreateOpeningOnProfile3DOb` takes a SddStiffener as input parameter and it returns created opening as output parameter in `ObjStrOpeningOnProfile`.

```
...
  Dim ObjStrOpeningOnProfile As StrOpeningOnProfile
  CreateOpeningOnProfile3DObject ObjSddStiffener, Ref3DObjectOpeningOnProfileDimensions, ObjStrOpeningOnProfile
...

...
Sub CreateOpeningOnProfile3DObject(iObjSddStiffener As SddStiffener, Ref3DObject As Reference, oObjStrOpeningOnProfile As StrOpeni

    'Add Opening On Profile
    AddOpening iObjSddStiffener, oObjStrOpeningOnProfile

   'Get StrOpening from StrOpeningOnProfile
    Dim ObjStrOpening As StrOpening
    Set ObjStrOpening = oObjStrOpeningOnProfile.StrOpening

    'Set opening type catStrOpeningMode3DObject for 3dobject opening
    ObjStrOpening.OpeningType = catStrOpeningMode3DObject

    'Set category
    Dim ObjStrCategoryMngt As StrCategoryMngt
    Set ObjStrCategoryMngt = ObjStrOpening.StrCategoryMngt
    ObjStrCategoryMngt.SetCategory "ManHole"

    'Set intersecting element for opening creation
    Dim ObjStrOpening3DObject As StrOpening3DObject
    Set ObjStrOpening3DObject = ObjStrOpening.StrOpening3DObject
    ObjStrOpening3DObject.IntersectingElement = Ref3DObject

    'Set forming mode and extrusion mode
    Set ObjStrOpeningExtrusionMngt = ObjStrOpening.StrOpeningExtrusionMngt
    ObjStrOpeningExtrusionMngt.ExtrusionMode = 1
    'ObjStrOpening.FormingMode = 0

    'Update the opening on profile object
    ObjPart.UpdateObject oObjStrOpeningOnProfile

End Sub
...

...
Sub AddOpening(iObjSddStiffener As SddStiffener, oObjSddOpeningOnProfile As StrOpeningOnProfile)
    Dim ObjStrOpeningsOnProfile As StrOpeningsOnProfile
    Set ObjStrOpeningsOnProfile = iObjSddStiffener.GetOpeningsOnProfile(0)
    'Add opening
    Set oObjSddOpeningOnProfile = ObjStrOpeningsOnProfile.Add

End Sub
...
```

Here get the `StrOpening` object from `StrOpeningOnProfile` and then set the various parameters to get the desired opening on profile.

8. **Removes opening on profile**

```
...
    Dim ObjStrOpeningsOnProfile As StrOpeningsOnProfile
    Set ObjStrOpeningsOnProfile = ObjSddStiffener.GetOpeningsOnProfile(0)
    ObjStrOpeningsOnProfile.Remove ObjStrOpeningOnProfile
...
```

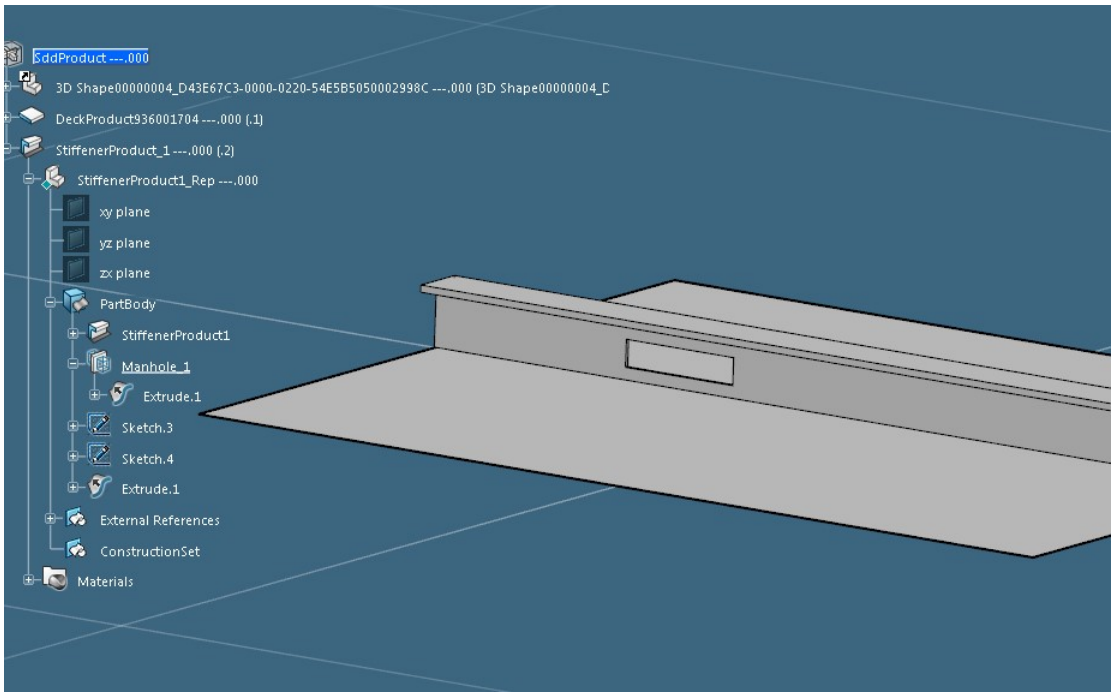
Object of `StrOpeningsOnProfile` is retrieved in `ObjStrOpeningsOnProfile`. Then on `ObjStrOpeningsOnProfile` object `Remove` method is called to remove th profile. This method takes a input parameter opening object to remove.

9. **Updates the Product Stiffener object**

```
...
  ObjPart.UpdateObject oObjStrOpeningOnProfile
End Sub
```

`UpdateObject` method updates the OpeningOnProfile object.

Fig.1: Profile opening using 3D object




# Testing SDD Collar

This use case primarily focuses on the methodology to test collar.

Related Topics
Structure Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcTestCollar.3dxml`, `CAAScdSddUcSteel_A42.3dxml`, `CAAScdSddUcWT18x179.5.3dxml`, `CAAScdSddUcRECT_RAN_WT18x179.5.3dxml`, `CAAScdSddUcCGR.3dxml` and `CAAScdSddUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\` where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcTestCollarSource.htm

This use case can be divided in 7 steps:

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Part object
3. Retrieves a SddProduct Plate object
4. Retrieves sdd slot object
5. Retrieves sdd collar object
6. Setting up collar parameters
7. Update the plate

1. **Searches and opens model which is named as "SddProduct"**

As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
Dim SDDPrdEditor  As  Editor
Dim prdName As String
prdName = "SddProduct"
OpenProduct prdName , SDDPrdEditor
...
```

The function **`OpenProduct`** returns **`SDDPrdEditor`**, a Editor object. After searching and opening of SDD model from underlying database the current active editor

is returned in **SDDPrdEditor**.

2. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Dim product1Service  As  PLMProductService
  Set product1Service = DDPrdEditor.GetService("PLMProductService")
  Dim ObjVPMRootOccurrence  As  VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference  As  VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances  As  VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

3. **Retrieves a SddProduct Plate object**

   In this step Uc retrieves the SddProduct Plate object.

```
...
  Set ListOfInstances = ObjVPMReference.Instances
  Set PlateRef = ListOfInstances.Item(1).ReferenceInstanceOf
  Set PlateRepInstances = PlateRef.RepInstances
  Set PlateRepInstReference = PlateRepInstances.Item(1).ReferenceInstanceOf
  Set PlatePart = PlateRepInstReference.GetItem("Part")

  Dim ObjSddProductPlate As SddProductPlate
  SFDProdSel.Add PlateRef
  Set ObjSddProductPlate = SFDProdSel.FindObject("CATIASddProductPlate")
  Dim ObjSddPlate As SddPlate
  Set ObjSddPlate = ObjSddProductPlate.SddPlate
...
```

   In this step Uc retrieves the SddProduct Plate as shown above.

4. **Retrieves sdd slot object**

   In this step UC finds a SDD slot.

```
...
  Dim ObjStrSlot As StrSlot
  GetSlot ObjSddPlate, ObjStrSlot
...

...
Sub GetSlot(iObjSddPlate As SddPlate, oObjStrSlot As StrSlot)

    Dim ObjStrSlots As StrSlots
    Set ObjStrSlots = iObjSddPlate.StrSlots
    'Get StrSlot
    Set oObjStrSlot = ObjStrSlots.Item(1)

End Sub
...
```

   In above lines, `GetSlot` method finds slot object from given plate. This slot has collar which is edited by modifying its properties.

5. **Retrieves sdd collar object**

   In this step UC finds a SDD collar on slot. This is the collar on which testing is to be done.

```
...
  Dim ObjStrCollar As StrCollar
  GetCollar ObjStrSlot, ObjStrCollar
...

...
Sub GetCollar(iObjStrSlot As StrSlot, oObjStrCollar As StrCollar)

   Dim ObjStrCollars As StrCollars
   Set ObjStrCollars = iObjStrSlot.Collars
   'Get StrCollar
   Set oObjStrCollar = ObjStrCollars.Item(1)

End Sub
...
```

   In above lines, `GetCollar` method finds collar object from given slot. This is the collar used for testing.

6. **Setting up collar parameters**

   In this step UC sets up collar parameters. Only public parameters can be modified.

```
...
  SetCollarParameters ObjStrCollar
...

...
Sub SetCollarParameters(iObjStrCollar As StrCollar)

   iObjStrCollar.Material = "Steel A42"

   iObjStrCollar.MaterialThrowOrientation = 0

   Dim oThickness As Parameter
   Set oThickness = iObjStrCollar.Thickness
```

```
    Dim oMaterialThrowOrientation As CATStrCollarThrowOrientation
    oMaterialThrowOrientation = iObjStrCollar.MaterialThrowOrientation

    Dim oMaterial As String
    oMaterial = iObjStrCollar.Material

    Dim ObjCollarParameters As StrParameters
    Set ObjCollarParameters = iObjStrCollar.CollarParameters
    Dim TOPC As Parameter
    Set TOPC = ObjCollarParameters.Item(2)
    TOPC.ValuateFromString("40mm")

    iObjStrCollar.Update

End Sub
...
```

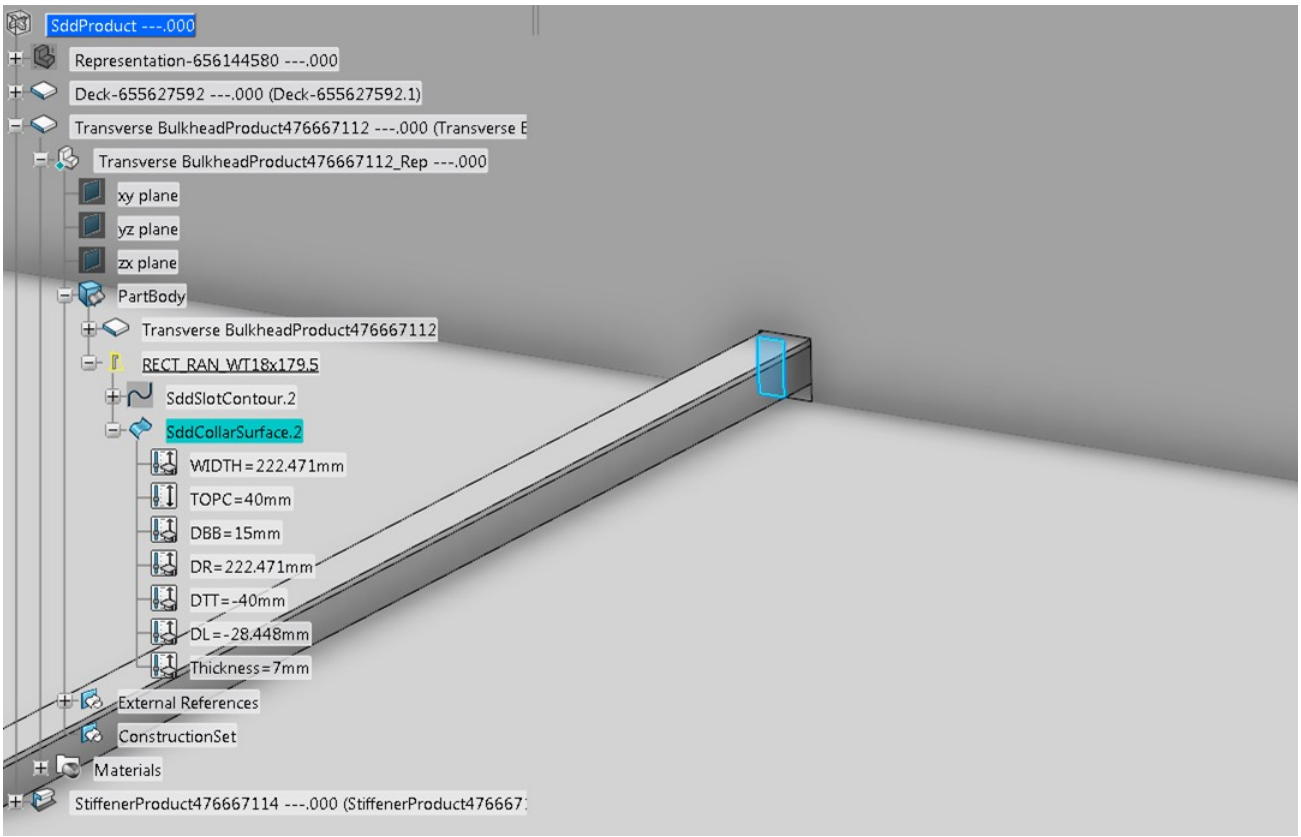
In above lines, `SetCollarParameters` method sets up different properties on collar. Here, `Material` property is used to get and set collar material. `Thickness` is used to get collar thickness. `MaterialThrowOrientation` is used to set and get different orientations of collar material (0 - catStrCollarThrowOrientationInvert, 1 - catStrCollarThrowOrientationNormal, 2 - catStrCollarThrowOrientationCentered). `CollarParameters` will get the public parameters on particular collar. Valuate it as required. `Update` updates the modified collar object.

7. **Update the plate**

```
...
    ObjSddProductPlate.Update
End Sub
```

Method `Update` updates the Product Plate.

Fig.1: Tested Collar




# Getting SDD Plate Sub-Elements

This use case primarily focuses on the methodology to get SDD plate sub elements.

Related Topics
Structure Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcGetPlateSubElements.3dxml`, `CAAScdSddUcSteel_A42.3dxml`, `CAAScdSddUcWT18x179_5.3dxml`, `CAAScdSddUcCGR.3dxml` and `CAAScdSddUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\`
  where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcGetPlateSubElementsSource.htm

This use case can be divided in 6 steps:

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Part object
3. Retrieves Service Manger (RfgService)
4. Retrieves a SddProduct Plate object
5. Get Plate Sub-Elements

6. Update the plate

1. **Searches and opens model which is named as "SddProduct"**

   As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim SDDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SDDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SDDPrdEditor`**, a Editor object. After searching and opening of SDD model from underlying database the current active editor is returned in **`SDDPrdEditor`**.

2. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Dim product1Service  As  PLMProductService
  Set product1Service = DDPrdEditor.GetService("PLMProductService")
  Dim ObjVPMRootOccurrence  As  VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference  As  VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances  As  VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

3. **Retrieves Service manager (RfgService)**

   In this step UC retrieves `RfgService`.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

   `GetService` method returns `RfgService`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

4. **Retrieves a SddProduct Plate object**

```
...
  Set ListOfInstances = ObjVPMReference.Instances
  Set PlateRef = ListOfInstances.Item(1).ReferenceInstanceOf
  Set PlateRepInstances = PlateRef.RepInstances
  Set PlateRepInstReference = PlateRepInstances.Item(1).ReferenceInstanceOf
  Set PlatePart = PlateRepInstReference.GetItem("Part")

  Dim ObjSddProductPlate As SddProductPlate
  SDDProdSel.Add PlateRef
  Set ObjSddProductPlate = SDDProdSel.FindObject("CATIASddProductPlate")
  Dim ObjSddPlate As SddPlate
  Set ObjSddPlate = ObjSddProductPlate.SddPlate
...
```

   In this step Uc retrieves the SddProduct Plate as shown above.

5. **Get Plate Sub-Elements**

   Now plate is available to get it's sub elements. Call `GetPlateSubElements` method to get plate's sub elements. `GetPlateSubElements` method takes a plate as input.

```
...
  Dim ObjSddPlateSubElementMngt As SddPlateSubElementMngt
  GetPlateSubElements ObjSddPlate, ObjSddPlateSubElementMngt
...

...
 Sub GetPlateSubElements(iObjSddPlate As SddPlate, oObjSddPlateSubElementMngt As SddPlateSubElementMngt)

  Set oObjSddPlateSubElementMngt = iObjSddPlate.PlateSubElements

  'Get Faces
  Dim oListOfFaces As References
  Set oListOfFaces = oObjSddPlateSubElementMngt.GetFaces(1)

 End Sub
...
```

   `GetFaces`: This method is used to retrieve the faces based on top and bottom face of plate. (catStrPlateFaceNameUndefined = 0, catStrPlateFaceBottom = 1, catStrPlateFaceTop = 2)

6. **Update the plate**

```
...
  ObjSddProductPlate.Update
End Sub
```

   Method `Update` updates the Product Plate.

# Getting SDD Profile Sub-Elements

This use case primarily focuses on the methodology to get SDD profile sub elements.

Related Topics
Structure Design Object Model Map
Launching an Automation Use Case

Before you begin: Note that:

- You should first launch CATIA and import the `CAAScdSddUcGetProfileSubElements.3dxml`, `CAAScdSddUcSteel_A42.3dxml`, `CAAScdSddUcWT18x179_5.3dxml`, `CAAScdSddUcCGR.3dxml` and `CAAScdSddUcSR.3dxml` files supplied in folder `InstallRootFolder\CAADoc\Doc\English\CAAScdSddSDDesign\samples\`
where `InstallRootFolder` is the folder where the CAA CD-ROM is installed.

Where to find the macro: CAAScdSddUcGetProfileSubElementsSource.htm

This use case can be divided in 6 steps:

1. Searches and opens model which is named as "SddProduct"
2. Retrieves Part object
3. Retrieves Service Manger (RfgService)
4. Retrieves a SddProduct Stiffener object
5. Get Profile Sub-Elements
6. Update the stiffener

1. **Searches and opens model which is named as "SddProduct"**

   As a first step, the UC retrieves a model "SddProduct" from DB and loads it and returns object of the Editor.

```
...
 Dim SDDPrdEditor  As  Editor
 Dim prdName As String
 prdName = "SddProduct"
 OpenProduct prdName , SDDPrdEditor
...
```

   The function **`OpenProduct`** returns **`SDDPrdEditor`**, a Editor object. After searching and opening of SDD model from underlying database the current active editor is returned in **`SDDPrdEditor`**.

2. **Retrieves Part object**

   In this step UC retrieves Part object ObjPart variable.

```
...
  Dim product1Service  As  PLMProductService
  Set product1Service = DDPrdEditor.GetService("PLMProductService")
  Dim ObjVPMRootOccurrence  As  VPMRootOccurrence
  Set ObjVPMRootOccurrence = product1Service.RootOccurrence
  Dim ObjVPMReference  As  VPMReference
  Set ObjVPMReference = ObjVPMRootOccurrence.ReferenceRootOccurrenceOf
  Dim ObjVPMRepInstances  As  VPMRepInstances
  Set ObjVPMRepInstances = ObjVPMReference.RepInstances
  Set ObjVPMRepReference = ObjVPMRepInstances.Item(1).ReferenceInstanceOf
  Set ObjPart = ObjPart = ObjVPMRepReference.GetItem("Part")
...
```

3. **Retrieves Service manager (RfgService)**

   In this step UC retrieves `RfgService`.

```
...
  Set Manager = CATIA.ActiveEditor.GetService("RfgService")
...
```

   `GetService` method returns `RfgService`. This service provides methods such GetReferencePlane, CreateProjectData, CreateRefSurfaceFeature.

4. **Retrieves a SddProduct Stiffener object**

```
...
  Set ListOfInstances = ObjVPMReference.Instances
  Set StiffenerRef = ListOfInstances.Item(2).ReferenceInstanceOf
  Set StiffenerRepInstances = StiffenerRef.RepInstances
  Set StiffenerRepInstReference = StiffenerRepInstances.Item(1).ReferenceInstanceOf
  Set StiffenerPart = StiffenerRepInstReference.GetItem("Part")

  Dim ObjSddProductStiffener As SddProductStiffener
  SDDProdSel.Add StiffenerRef
  Set ObjSddProductStiffener = SDDProdSel.FindObject("CATIASddProductStiffener")
  Dim ObjSddStiffener As SddStiffener
  Set ObjSddStiffener = ObjSddProductStiffener.SddStiffener
...
```

   In this step Uc retrieves the SddProduct Stiffener as shown above.

5. **Get Profile Sub-Elements**

   Now profile is available to get it's sub elements. Call `GetProfileSubElements` method to get profile's sub elements. `GetProfileSubElements` method takes any profile as input.

```
...
  Dim ObjStrProfileSubElementMngt As StrProfileSubElementMngt
  GetProfileSubElements ObjSddStiffener, ObjStrProfileSubElementMngt
...

...
Sub GetProfileSubElements(iObjSddStiffener As SddStiffener, oObjStrProfileSubElementMngt As StrProfileSubElementMngt)

  Set oObjStrProfileSubElementMngt = iObjSddStiffener.ProfileSubElements

  'Get Faces
  Dim oListOfFaces As References
  Set oListOfFaces = oObjStrPlateSubElementMngt.GetFaces("WebInner+")
```

```
  'Get Edges
  Dim oListOfEdges As References
  Set oListOfEdges = oObjStrProfileSubElementMngt.GetEdges("Start", "WebInner-")

End Sub
...
```

`GetFaces`: This method is used to retrieve any set of faces corresponding to profile face name.

`GetEdges`: This method is used to retrieve the edges corresponding to the intersection of two adjacent faces.

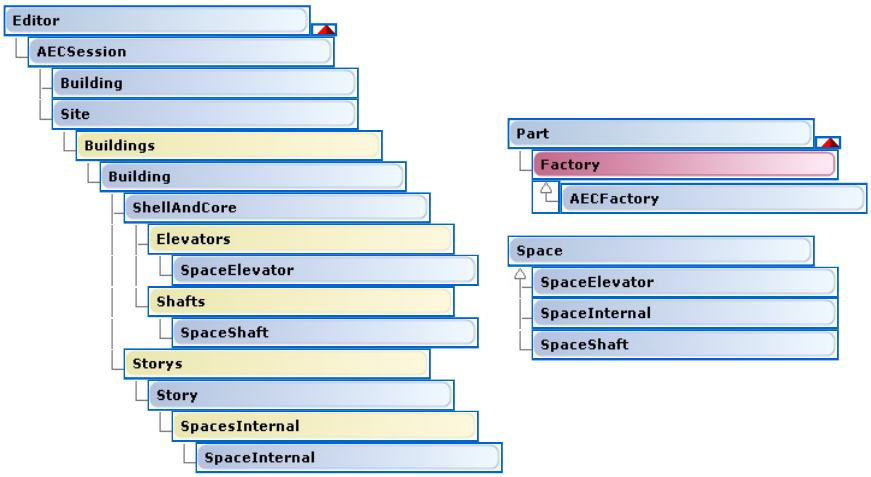
6. **Update the stiffener**

```
...
  ObjSddProductStiffener.Update
End Sub
```

Method `Update` updates the Product Stiffener.

# Buildings Space Planning Object Model Map

See Also Legend




Use the **GetItem** method of the **Editor** object to retrieve the **AECSession** object.

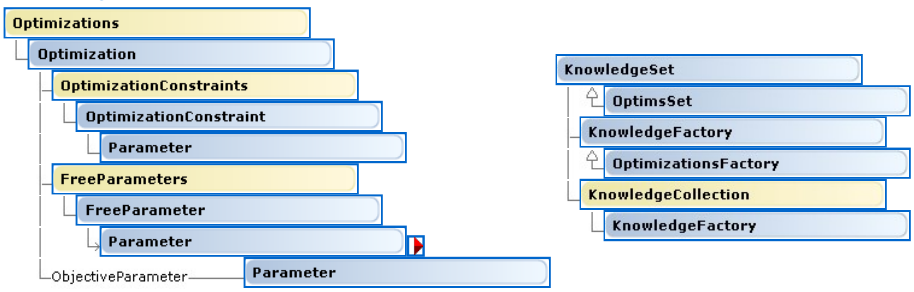
```
Dim oAECSession  As AECSession
oAECSession = CATIA.ActiveEditor.GetItem("SWXBuAECSession")
...
```

Use the **GetItem** method of the **Part** object to retrieve the **AECFactory** object.

```
...
Dim oAECFactory  As AECFactory
oAECFactory = oPart.GetItem("SWXBuAECFactory")
```

# Design Optimization Object Model Map

See Also Legend




Each **Optimization** object aims to minimize or maximize the value of an objective parameter by modifying the values other parameters, called *free parameters*, with respect to constraints. A constraint is a relationship between parameters equivalent to one of the following expressions:

```
foo(param1, param2, ...) < constantValue
```

```
foo(param1, param2, ...) = constantValue
```

Those constraints are seen as an aggregated collection of **OptimizationConstraint** objects.

Free parameters are represented by an aggregated collection of **FreeParameter** objects pointing to the parameter itself. It allows you to specify a step and a range to monitor the variations of the parameter value.

# Knowledge Basics Overview

This article provides the basis for understanding how to create macros using knowledgeware features. These knowledgeware features are supplied in the Infrastructure product (parameters, formulas and design tables) as well as in the Knowledge Basics product (rules and checks).

The Knowledge Basics Automation objects are those whereby you can create and manipulate the knowledgeware features in a script. These Automation objects can be divided into two categories:

1. The objects which provide you with the creation methods. They are the **RelsSet** and **ParmsSet** objects. These objects are only implemented on product representation references (3D shape and technological representation references).
2. The knowledgeware objects whereby you can manipulate the Parameter, Formula, Rule, Check and Design Table features once they have been created. The hierarchy of these objects is described in Parameters Object Model Map and Relations Object Model Map.

The Use Cases provided as samples are fully commented and should help you understand how to proceed to write simple macros as well as fully-fledged macros manipulating knowledgeware features.

Warning: Knowledgeware features are inside streams pointed by PLM Representation References. If a stream is not fully loaded in session, or if a PLM Representation Reference is not loaded in session, you could not reach existing Knowledgeware features.

## Creating Knowledgeware Objects

The entry point is the **RelsSet** or **ParmsSet** objects. To retrieve the appropriate object, you must retrieve the root object of the product representation reference. This object aggregates all the objects making up the representation reference: for instance the bodies, the relations, the parameters to name but a few. Mull over this object properties, they all provide you with collections or factories to create objects.

For example, you can retrieve a **ParmsSet** object that way:

```
Dim oActivePart As VPMRepReference
Set oActivePart = CATIA.ActiveEditor.ActiveObject

Dim oKnowledgeObjects As KnowledgeObjects
Set oKnowledgeObjects = oActivePart.GetItem("KnowledgeObjects")

Dim oParams As ParmsSet
Set oParams = oKnowledgeObjects.GetKnowledgeRootSet(True, 0) ' kweParametersType = 0
```

The **GetItem** method called against the active **Part** object using the `KnowledgeObjects` arguments returns a KnowledgeObjects object, the **GetKnowledgeRootSet** method of which creates the requested collection object and returns it.

Using `False` instead of `True` as the first argument of the **GetKnowledgeRootSet** method would have returned the existing **ParmsSet** object, if any, instead of creating it. Using, for example, `1 (kweRelationsType)` instead of `0 (kweParametersType)` would have created a **RelsSet** collection object instead of a **ParmsSet** collection object.

But prior to writing this, you must check that your object is a 3D shape.

To create an object, you just have to use the appropriate Create*xxx* method.

```
Dim oSphereRadius As Parameter
Set oSphereRadius = oParams.Factory.CreateInteger("StringLength",0)
```

## Scanning the Knowledgeware Objects

**ParmsSet** as well as **RelsSet** can be scanned through their **Collection** attribute. An item is retrieved from its collection using the **Item** method and the index of the item in the collection. Usually, the argument representing the index in the **Item** method is a Variant. This means that it can either represent the rank of the item in the collection or the name you assigned to this item using the **Name** property. The rank in a collection begins at 1. For example, assume that the Parameter named "StringLength" is the sixth parameter in the **ParmsSet** collection. To retrieve this parameter from the `oParams` object, write:

```
Dim oParameter1 As Parameter
Set oParameter1 = oParams.Collection.Item(6)
```

or write:

```
Dim oParameter1 As Parameter
Set oParameter1 = oParams.Collection.Item("StringLength")
```

Each collection has a **Count** property that holds the number of items in the collection. This is a handy property to scan the whole collection. You should use the **For each** structure for performance.

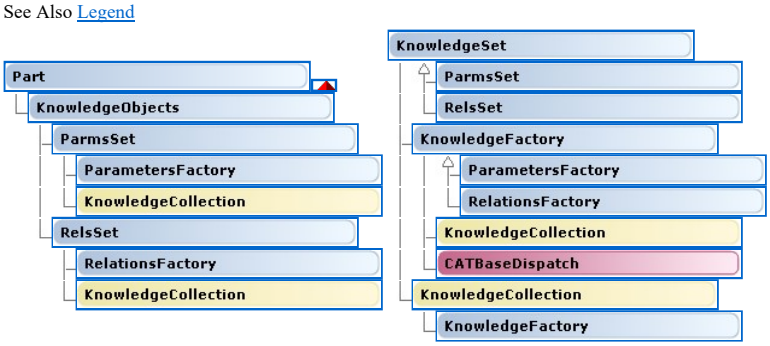
Warning: When counting the items in a **Parameters** collection object, mind that you count all the representation parameters and not only the user parameters.

## Manipulating Knowledgeware Objects

After they have been created, knowledgeware objects are managed through their properties and methods. To find out what they are, see the Knowledge Basics Automation API Reference.

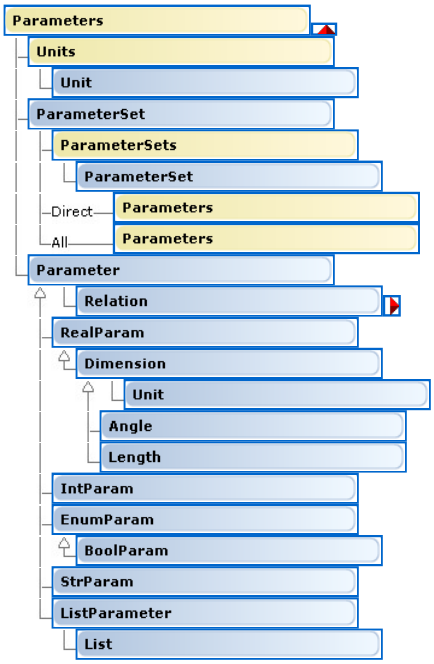
# Knowledge Basics Object Model Map

See Also Legend



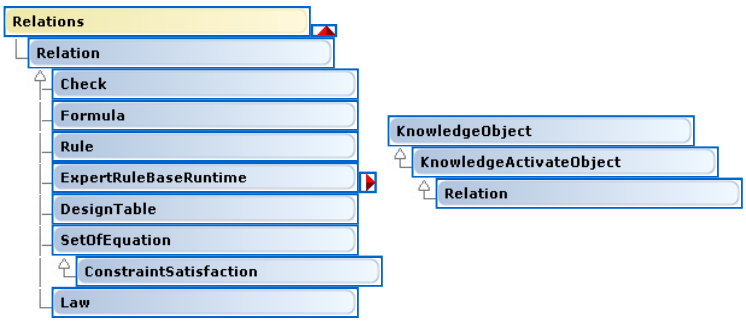
# Parameters Object Model Map

See Also Legend




# Relations Object Model Map

See Also Legend




# Creating Parameters and Relations

This use case fundamentally illustrates creating different types of Parameters and Relations between them

---

Before you begin: Note that:

- Launch CATIA
- In Tools-> Options-> Infrastructure-> 3D Shape Infrastructure -> Display Tab select Parameters and Relations

Related Topics
*<topic1>*
*<topic2>*

Where to find the macro: CAAScdKniUcCreateParametersAndRelationsSource.htm

This use case can be divided in 8 steps

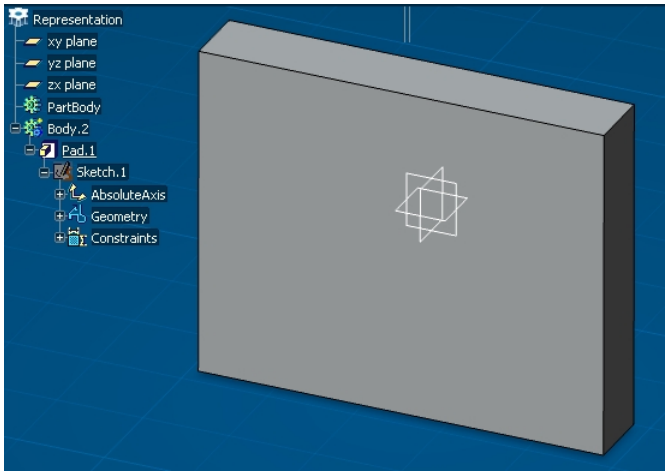
1. Creates the 3DShape
2. Retrieves the KnowledgeObjects Object from Part
3. Creates the Parameters Set beneath the Part
4. Creates the Relations Set beneath the Part
5. Creates the Parameter Objects
6. Creates the Relation Objects
7. Updates the Part Object
8. Validates the Relations

1. **Creates the 3DShape**

```
...
    Dim MyNewPart As Part
    Create3DShape MyNewPart
...
```

The `Create3DShape` sub routine creates a new 3DShape with geometry as shown in the following picture.

Fig. 1 Creating 3D Shape




*MyNewPart* is the Part Object of the 3D Shape, the object aggregating all the objects making up the 3D shape.

2. **Retrieves the Knowledge Objects Object from Part**

```
...
    Dim oKnowledgeObjects As KnowledgeObjects
    Set oKnowledgeObjects =  MyNewPart.GetItem("KnowledgeObjects")
...
```

The *GetItem* method called against the active Part object using the *KnowledgeObjects* arguments returns a *KnowledgeObjects* object.

3. **Creates the Parameters Set beneath the Part**

The *GetKnowledgeRootSet* method of *KnowledgeObjects* creates the requested collection object and returns it.

Creates the Parameters Set

```
...
    Dim  oParmsSet As ParmsSet

    Set oParmsSet = oKnowledgeObjects.GetKnowledgeRootSet(True, kweParametersType)
...
```

The *Parameters Set* is the feature Parameters beneath the *Part* feature (see Fig.2). To create it, we use the *GetKnowledgeRootSet* method of the *KnowledgeObjects* object using TRUE as first argument. The second argument specifies the type of set to create. *kweParametersType* it is the *ParmsSet* object (representing the Parameters feature).

Note: About the first argument of *GetKnowledgeRootSet* : If the two features already exist, the TRUE value as argument is equivalent to FALSE (it makes a retrieval)

4. **Creates the Relations Set beneath the Part**

Creates the Relations Set

```
...
    Dim  oRelsSet As RelsSet

    Set oRelsSet = oKnowledgeObjects.GetKnowledgeRootSet(True, kweRelationsType)
...
```

The *Relations set* is the feature *Relations* beneath the *Part* feature (see Fig.2). To create it, we use the *GetKnowledgeRootSet* method of the *KnowledgeObjects* object using TRUE as first argument. The second argument specifies the type of set to create.*kweRelationsType* it is the *RelsSet* object (representing the Relations feature).

5. **Creates the Parameter Objects**

```
...
    Dim oParametersFactory As ParametersFactory
    Set oParametersFactory = oParmsSet.ParametersFactory

    Dim oParamString1 As Parameter
    Set oParamString1 = oParametersFactory.CreateString("ProjectId", "CATLifKnowledgeAdvisor")

    Dim  oParamSphereVol As Parameter
    Set oParamSphereVol = oParametersFactory.CreateDimension("SphereVolume", "VOLUME", 0)
...
```

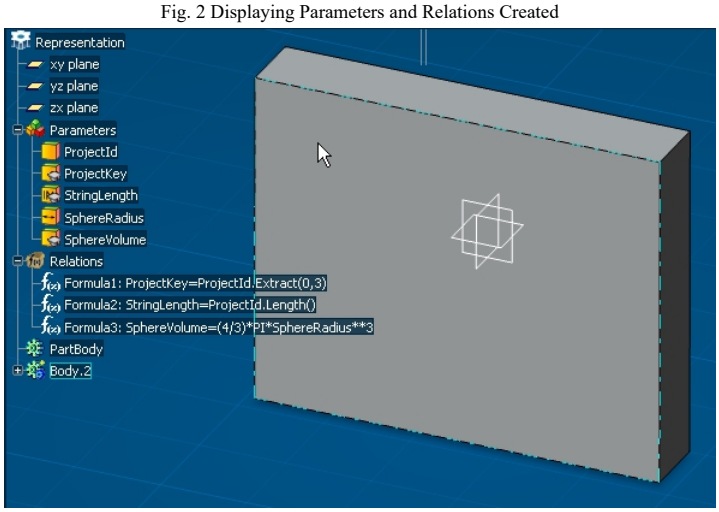
The *ParametersFactory* is retrieved from the *ParametersSet* object, different *Parameter* types are created using appropriate *Createxxx* methods of the *ParametersFactory*

6. **Creates the Relation Objects**

```
...
    Dim oRelationsFactory As RelationsFactory
    Set oRelationsFactory = oRelsSet.ParametersFactory

    Dim oFormula1 As Formula
    Set oFormula1 = oRelationsFactory.CreateFormula("Formula1", "", oParamString2, "ProjectId.Extract(0,3)")

    Dim oFormula4 As Formula
    Set oFormula4 = oRelationsFactory.CreateFormula("Formula4", "", oParamSphereVol, "(4/3)*PI*SphereRadius**3")
...
```

The *RelationsFactory* is retrieved from the *RelationsSet* object, different *Relations* are created using *CreateFormula* method of the *RelationsFactory*

Fig. 2 Displaying Parameters and Relations Created




7. **Updates the Part Object**

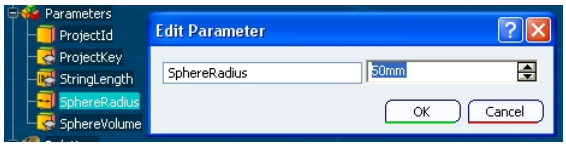
```
...
    MyNewPart.Update
...
```

Updates the Part object, *MyNewPart* , result with respect to its specifications

8. **Validates the Relations**

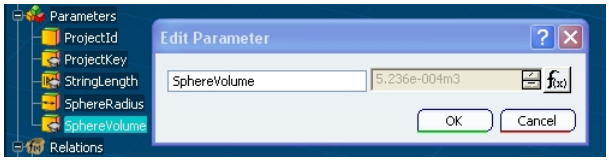
The created *Parameter* names are displayed in specification tree, from it edit *SphereRadius* parameter value.

Fig. 6 Edit Parameter




The *SphereVolume* parameter value changes according to Formula3 *Relation* created by macro.

Fig. 7 Check Parameter




Similarly we can check all other *Relations* applied on created *Parameters*.

# Scanning the Parameter Set for the hidden Parameters

This use case retrieves the parameter set created just below the Part feature. This parameter set is also named the root parameter set. Once retrieved, it is scanned to list all (i.e. recursively) its hidden boolean parameters.

Related Topics
*<topic1>*
*<topic2>*

Before you begin: Note that:

- Launch CATIA
- In Tools->Options->Infrastructure->3D Shape Infrastructure->Display Tab, select Parameters and Relations

Where to find the macro: CAAScdKniUcScanHiddenParametersSource.htm

This use case can be divided in 7 steps

1. Creates 3DShape with a Parameter Set
2. Retrieves the Knowledge Service
3. Retrieves the Part Knowledge Collection
4. Retrieves the Created Parameter Set
5. Retrieves its Parameter Collection
6. Scans Parameter Collection for the hidden Boolean Parameters
7. Displays the list of the hidden Boolean Parameters

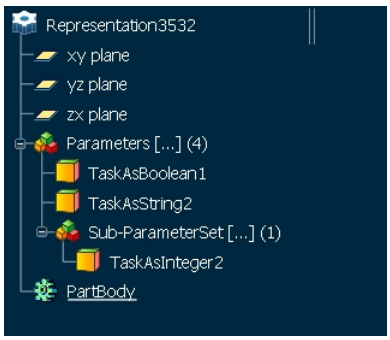
1. **Creates 3DShape with a Parameter Set**

```
...
    Dim MyNewPart As Part
    Create3DShape MyNewPart
```

```
...
```

The `Create3DShape` sub routine creates a new 3DShape with the parameters residing under a newly created parameter set as shown in the following figure. Here which 4 are hidden, as shown in the following figure, only 2 parameters are visible. Some parameters are string like others are boolean.

Fig. 1 Creating a 3D Shape with the Parameters




`MyNewPart` is the Part object of the 3DShape, the object aggregating all the objects making up the 3DShape.

2. **Retrieves the Knowledge Service**

```
...
    Dim oKnowledgeServices As KnowledgeServices
    Set oKnowledgeServices = CATIA.GetSessionService("KnowledgeServices")
...
```

The *`GetSessionService`* method retrieves the knowledge service from the current `CATIA` session.

3. **Retrieves the Part Knowledge Collection**

```
...
    Dim oKnowledgeCollectionForPart As KnowledgeCollection
    Set oKnowledgeCollectionForPart = oKnowledgeServices.GetKnowledgeCollection(MyNewPart, kweParametersType)
...
```

The method *`GetKnowledgeCollection`* retrieves the KnowledgeCollection object associated with the Part feature (`MyNewPart`). The second parameter of this met the returned collection will contain as direct childrens, all parameter sets and parameters below the Part feature.

4. **Retrieves the Created Parameter Set**

```
...
    Dim oParmsSetForMyParameterSet As ParmsSet
    Set oParmsSetForMyParameterSet = oKnowledgeCollectionForPart.Item(1)
...
```

When using the method *`Item`* on a KnowledgeCollection object created with the `kweParametersType` parameter, you retrieve either a parameter set or a paramet can be an index or a name. In our case, since below the Part feature the first element is a parameter set, this piece of code retrieves the parameter set created in the hidden and visible parameters.

5. **Retrieves its Parameter Collection**

The objective of this section is to retrieve all the parameters contained by our parameter set.

```
...
    Dim oKnowledgeCollection As KnowledgeCollection
    Set oKnowledgeCollection = oParmsSetForMyParameterSet.Collection
...
```

First we retrieve as a KnowledgeCollection object, `oKnowledgeCollection` , the collection to scan oKnowledgeCollectionForPart for any kind knowledge objects this KnowledgeCollection object.

```
...
    Dim oParamCollection As KnowledgeCollection
    Set oParamCollection = oKnowledgeCollection.Find(kweParameterObjectType, True)
...
```

Finally, using the `Find` method we are able to create a collection whose the contents is depending on the method's argument. `kweParameterObjectType` means th parameter objects, and `True` means that the search is recursive.

6. **Scans the Parameter Collection for the hidden Boolean Parameters**

The Use Case scans the hidden parameters of type `Boolean` from the retrieved collection.

```
...
    Dim  i As Integer
    For i = 1 To oParamCollection.Count
        Dim oParam As Parameter
        Set oParam = oParamCollection.Item(i)
        Dim  HiddenNumber As Integer
        If (oParam.Hidden) Then
            If TypeName(oParam) = "BoolParam" Then
                oParamString1 = oParamString1 & vbCrLf & oParam.Name & "   " & oParam.Value & "   " & "'" & oParam.Comment & "'":
...
```

Test whether the value returned by the *`Hidden`* property of the parameter is "True" and then check whether the `Parameter` belongs to `BoolParam` class object. If y variable.

Here we are scanning the hidden parameter list only for the boolean type of parameters. Here we are using the function *`TypeName`* to check that the parameter is of

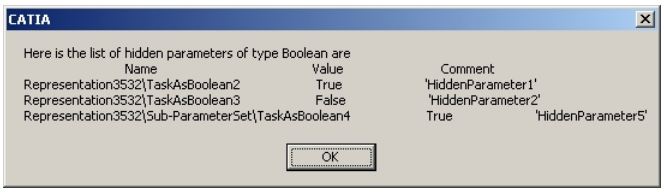
representing boolean parameter.

7. **Displays the list of hidden Boolean Parameters**

```
...
    If (HiddenNumber > 0) Then
    MsgBox oParamString1
...
```

The hidden boolean parameter names, values and comments are displayed in the message box.

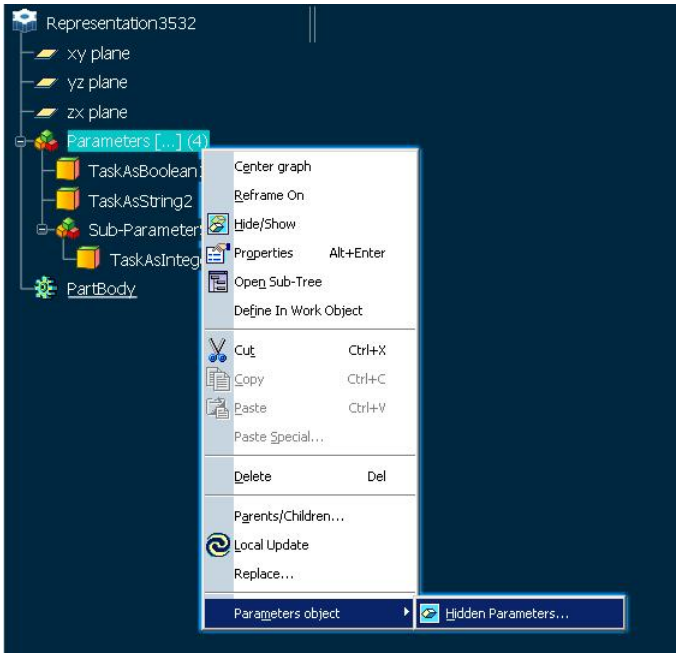
Fig. 2 Displaying the Hidden Boolean Parameters




You can modify interactively the status of the parameters and see the hidden parameters in the structure as described below

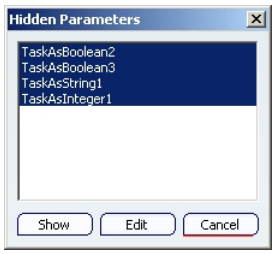
In the specification tree Right-Click on the Parameters. From the contextual menu select the Parameters Object->Hidden Parameters.

Fig. 3 Displaying all the Hidden Parameters Interactively



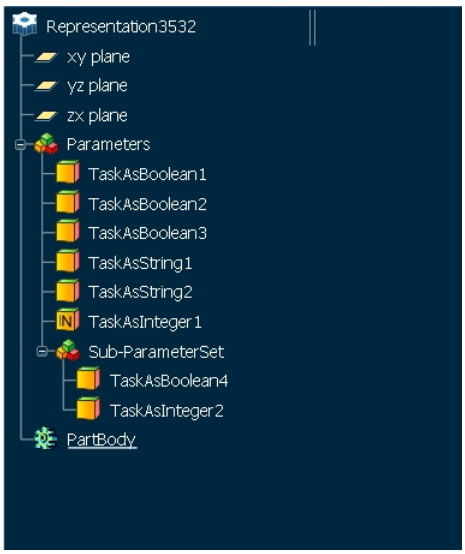

Select the parameters which you want to display and click on the Show button.

Fig. 4 Showing the Hidden Parameters




All the parameters are displayed in the specification tree as shown below.

Fig. 5 Displaying All the Parameters




# Know-how Reuse Overview

This article provides the basis for understanding how to create macros replaying most of the operations you can perform with the Know-how Reuse app. Note that the Know-how Reuse app does not provide you with journaling capabilities. The objects described below as well as the Use Cases which illustrate how to manipulate these objects can only be used to write a VB macro to be replayed later on.

Basically, to start with Know-how Reuse Automation, you need know:

1. How to create the Know-how Reuse objects you are going to manipulate. These objects are the rule base, the rule sets, the expert rules and the expert checks. These objects can be created from the **RuleBasesSet** object
2. How to manipulate these objects
3. How to launch some Know-how Reuse specific operations such as solving a rule base, generating a check report, or highlighting the elements that don't fulfill the criteria specified in a check.

The Use Cases provided as samples are fully commented and should help you understand how to proceed to write simple macros as well as fully-fledged macros manipulating Know-how Reuse objects.

## Creating a Rule Base

The entry point is the **RuleBasesSet** object which provides you with the factory and its **CreateRuleBase** method. This method takes as its argument the rule base name.

```
Dim oRBSet As ExpertRuleBasesSet
Set oRBSet = CATIA.ActiveEditor.ActiveObject.GetItem("KnowledgeObjects").GetKnowledgeRootSet(True, 3) ' kweRuleBasesType = 3

' Create the RB1 rulebase
Dim oRuleBase As ExpertRulebase
Set oRuleBase = oRBSet.Factory.CreateRuleBase("RB1")

' Add a root of facts to the rulebase
aRuleBase.AddRootOfFacts CATIA.ActiveEditor.ActiveObject
```

The **GetItem** method called against the active object object using the `KnowledgeObjects` arguments returns a KnowledgeObjects object, the **GetKnowledgeRootSet** method of which creates the requested collection object and returns it.

Using `False` instead of `True` as the first argument of the **GetKnowledgeRootSet** method would have returned the existing **RuleBasesSet** object, if any, instead of creating it.

After a rule base has just been created, it is in a to-be-solved status.

## Creating a Rule Set

In interactive mode, you don't have to create rule sets. They are automatically created upon creation of rules and checks. It is the same in VB.

In the present version, rule sets cannot be gathered in a collection. There are no direct means to retrieve the set of **RuleSet** objects.

## Creating Expert Rules and Expert Checks

Prior to using the rule or check creation methods, you need retrieve the **RuleSet** object from the rule base. To create an expert rule or expert check, you must specify the rule set name as the last argument of the **CreateRule** or **CreateCheck** method. In the statements below, `RuleSet.1` is created if it does not exist yet. If it already exists, the `HDiaCheck` is created right below it.

```
Dim oCheck As ExpertCheck
Set oCheck= oRuleBase.RuleSet.CreateCheck("HDiaCheck","H:Hole","H.Diameter>12mm","RuleSet.1")
```

## Determining whether a Rule Base is to Be Solved

The status of a rule base is described by its FingerPrint attribute. If it is set to 0, the rule base must be solved, if it is set to 1, the rule base is solved.

## Manipulating the Components of a Rule Base

The experts rules and checks making up a rule base are grouped into an **ExpertRuleBaseComponentRunTimes** collection object. The component type where by you can differentiate a rule from a check in a collection is **ExpertRuleRunTime** or **ExpertCheckRunTime**.

### Know-how Reuse Specific Operations

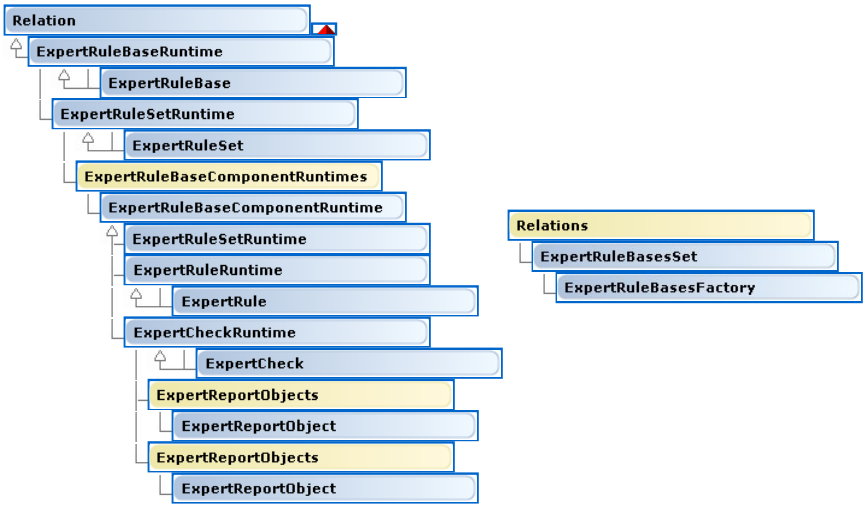
To solve a rule base, use the *rulebase*.Deduce() method.

To generate a report, use the *rulebase*.Report() method

To highlight the elements which do not satisfy the checks, use the *rulebase*.Highlight() method.

# Know-how Reuse Object Model Map

See Also Legend




**ExpertRuleBase**, **ExpertRuleSet**, **ExpertRule**, and **ExpertCheck** objects are accessible through two kinds of objects:

1. Run time objects that contain information relative to the use of the object in the context, for example the result of an **ExpertCheck** object applied to the current 3D shape representation
2. Definition objects that contain the definition of the object, for example the body of an **ExpertCheck** object.

Each run time object allows you to access its definition object through an **xxEdition** property. For example, the **ExpertRuleBaseRuntime** object has a **RuleBaseEdition** property that returns the associated **ExpertRuleBase** object.

An **ExpertRuleBaseRuntime** object and its associated **ExpertRuleBase** object are created by the **CreateRuleBase** method on the **Factory** attribute of the **RuleBasesSet** object. It recursively aggregates all other Knowledge Expert objects.

The **ExpertRuleSetRuntime**, **ExpertRuleRuntime** and **ExpertCheckRuntime** objects inherit from the **ExpertRuleBaseComponentRuntime** object.

An **ExpertRuleSet** object that aggregates a collection of **ExpertRuleBaseComponentRuntime** objects is so able to aggregate other **ExpertRuleSet** objects as well as **ExpertRuleRuntime** objects and **ExpertCheckRuntime** objects that are all created using its **CreateCheck**, **CreateRule** and **CreateRuleSet** methods.

An **ExpertCheckRuntime** object aggregates the result of its application to the current 3D shape representation as two collections of **ExpertReportObject** objects. The first one contains the combinations of objects for which the check failed, and the other one the combinations of objects for which the check succeeded.

# Creating and Solving Know-how Reuse Rule Base

This use case fundamentally illustrates creating and solving a rule base.

---

Before you begin: Note that:

- Launch CATIA
- In Tools->Options->Infrastructure->3D Shape Infrastructure -> Display Tab select Parameters and Relations.

Related Topics
*<topic1>*
*<topic2>*

Where to find the macro: CAAScdKhwUcCreateAndSolveRuleBaseSource.htm

This use case can be divided in 10 steps:

1. Creates a 3DShape with Holes
2. Retrieves the KnowledgeObjects Object from the Part
3. Creates a Set of Rule Base
4. Creates a Rule Base
5. Adds the Part Feature as the Root of Facts to the Rule Base
6. Creates Expert Checks for the Rule Base
7. Determines whether the Rule Base to Be Solved or Not
8. Solves the Rule Base if the Rule Base is to Be Solved
9. Highlights the Failing Items
10. Updates the Part Object

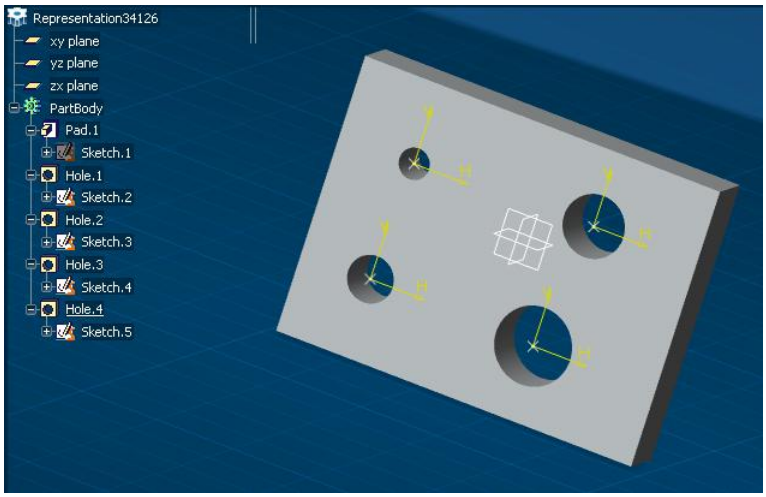
1. **Creates a 3DShape with Holes**

```
...
    Dim MyNewPart As Part
    Create3DShape MyNewPart
...
```

The `Create3DShape` sub routine creates a new 3DShape with geometry as shown in the following picture. This is a pad having four Holes of diameter 10, 15, 20

and 25 mm.

Fig. 1 Displaying the 3DShape




`MyNewPart` is the Part Object of the 3D Shape, the object aggregating all the objects making up the 3D shape.

2. **Retrieves the KnowledgeObjects Object from the Part**

```
...
    Dim oKnowledgeObjects As KnowledgeObjects
    Set oKnowledgeObjects = MyNewPart.GetItem("KnowledgeObjects")
...
```

The `GetItem` method called against the Part object using the `KnowledgeObjects` arguments returns a **KnowledgeObjects** object.

3. **Creates a Set of Rule Base**

```
...
    Dim oExpertRuleBasesSet As ExpertRuleBasesSet
    Set oExpertRuleBasesSet = MyNewPart.GetKnowledgeRootSet(True, 3)
...
```

Fig. 2 shows result of above call. In this figure the RuleBases below the Relations is a set of rule base created by above call. In code, a set of rule base is an **ExpertRuleBasesSet** object. In context of a Part feature this set is always aggregated under a **Relations** object. So, the **Relations** object is created automatically if it is not created earlier. To create an **ExpertRuleBasesSet** object we use the `GetKnowledgeRootSet` method of the **KnowledgeObjects** object using TRUE as first argument. The second argument specifies type of the `KnowledgeSet` to create. Its value is 3 if type of the `KnowledgeSet` to create is an **ExpertRuleBasesSet** object.

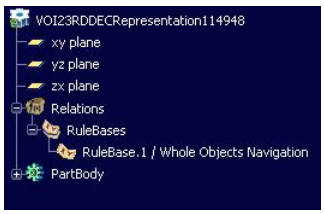
Fig. 2 Displaying the Set of Rule Base under the Relations


4. **Creates a Rule Base**

```
...
    Dim oExpertRuleBasesFactory As ExpertRuleBasesFactory
    Set oExpertRuleBasesFactory = oExpertRuleBasesSet.Factory
    Dim oExpertRuleBase As ExpertRuleBase
    Set oExpertRuleBase = oExpertRuleBasesFactory.CreateRuleBase("RuleBase.1")
...
```

Fig. 3 shows result of above call. In this figure the RuleBase.1 is the the rule base created by above call. In code, a rule base is an **ExpertRuleBase** object. This object is created from the factory of a rule base. In code, a factory is an **ExpertRuleBasesFactory** object. The factory is retrieved from an **ExpertRuleBasesSet** object and uses the `CreateRuleBase` function to create an **ExpertRuleBase** object. The argument is name of the **ExpertRuleBase** object to be created. Same name appears in specification tree.

Fig. 3 Displaying Rule Base under Set of Rule Base



5. **Adds the Part Feature as the Root of Facts to the Rule Base**

```
...
    oRuleBase.AddRootOfFacts MyNewPart
...
```

In the Know-how Reuse terminology the `Fact` is the set of object provided as input to solver. This `Fact` is found by solver through a `Root` object. While interactively the `Root` is associated with solver automatically, in code this is done with the `AddRootOfFacts` function of the `ExpertRuleBase` object.

6. **Creates Expert Checks for the Rule Base**

An expert checks is related to the rule base via a rule sets. So, in first step rule sets is retrieved from the rule base created in previous step. Then, in second step the expert checks is created from the rule sets.

1. Retrieves the Rule Sets from the Rule Base

```
...
    Dim oExpertRuleSet As ExpertRuleSet
    Set oExpertRuleSet = oExpertRuleBase.RuleSet
...
```

A rule base is linked with a rule sets. In code, a rule sets is **ExpertRuleSet** object which is retrieved using the `RuleSet` property of the **ExpertRuleBase** object.

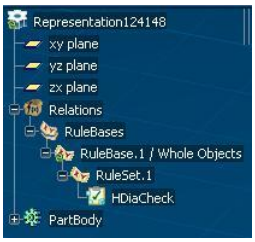
2. Creates the Expert Checks

```
...
    Dim oExpertCheck As ExpertCheck
    Set oExpertCheck = oExpertRuleSet.CreateCheck("HDiaCheck", "Hole:Hole", "Hole.Diameter>12mm", "RuleSet.1")
...
```

In code, the expert checks is an **ExpertCheck** object. This object is created from the `CreateCheck` method of the **ExpertRuleSet** object.

Fig. 4 shows result of above calls. First argument in the `CreateCheck` method is name of the **ExpertCheck** object to be created. Same name appears in specification tree. Second argument assigns a variable name Hole to **Hole** object which are part of representation created in Step 1 . When this check is solved, the solver replace Hole by actual **Hole** objects. Third argument defines the check to be made on object specified in argument 2. Here the expert checks check if diameter of holes are greater than 12 mm.

Fig. 4 Displaying the Rule Set and the Check under the Rule Base




7. **Determines whether the Rule Base to Be Solved or Not**

```
...
    If (oExpertRuleBase.Fingerprint = 0)
...
```

The `Fingerprint` value determines if an `ExpertRuleBase` is already solved. The `Fingerprint` value is 0 if an `ExpertRuleBase` is unsolved and it is 1 if it is already solved.

8. **Solves the Rule Base if the Rule Base is to Be Solved**

```
...
    oExpertRuleBase.Deduce
...
```

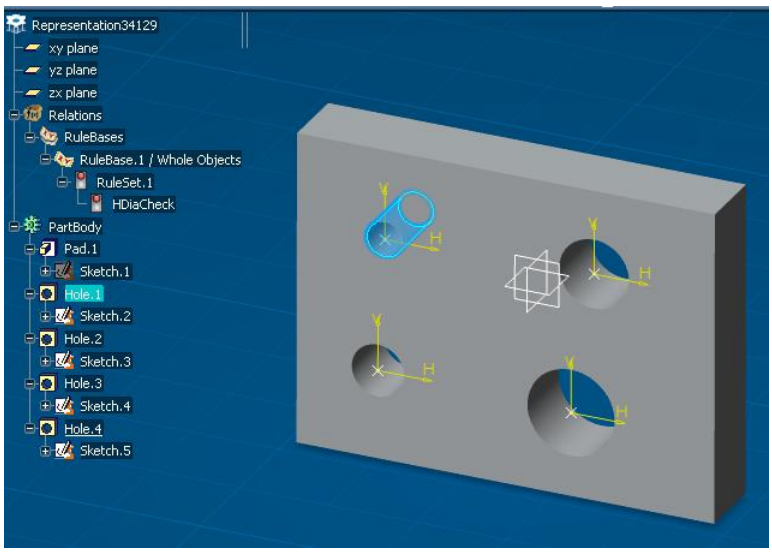
The `Deduce` method solves the `ExpertRuleBase`.

9. **Highlights the Failing Items**

```
...
    oExpertCheck.Highlight
...
```

The `Highlight` method of the **ExpertCheck** object highlights the failed items. Here Hole having diameter less than 12mm is highlighted as shown in Fig. 5. Failed items get highlighted in geometry area as well as in specification tree. Note that expert checks as defined in third argument of the `CreateCheck` method in step 6 is to check if diameter of holes are greater than 12 mm.

Fig. 5 Displaying Highlighted Items for which Check Failed



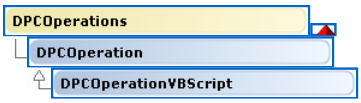

10. **Updates the Part Object**

```
...
    MyNewPart.Update
...
```

Updates the Part Object, `MyNewPart` , result with respect to its specifications

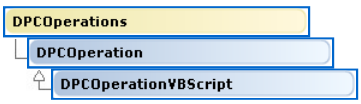
# Know-how Apps Components Object Model Map

See Also Legend




# DPCOperations Collection Object

See Also Legend Use Cases Properties Methods




Represents the collection of operations associated with the object of product, a feature, or a 3D shape representation.

The **DPCOperations** collection object can be retrieved from any object interactively extended by the Automated Design Process, such as a product, a **Part** object, or a **Body** object.

Use the **GetItem** method of the active object, here a **Part** object interactively extended by the Automated Design Process to return the **DPCOperations** collection object.

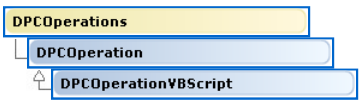
```
Dim oRoot As Object
Set oRoot = CATIA.ActiveEditor.ActiveObject
Dim cOperations As DPCOperations
Set cOperations = oRoot.GetItem("CATGetDPCOperations")
```

Use the **Operate** method of the **DPCOperations** collection object to retrieve and run an operation, for example the operation named `MyOperation`.

```
Dim oOperation As DPCOperation
Set oOperation = cOperations.Operate("MyOperation")
```

# DPCOperationVBScript Object

See Also Legend Use Cases Properties Methods




Represents a VB Script operation.

The methods of a VB Script operation are intended to be used in a macro launched by the object itself. They enable you to retrieve, test, and possibly set the values of the variables used as input or output, or of the internal variables.