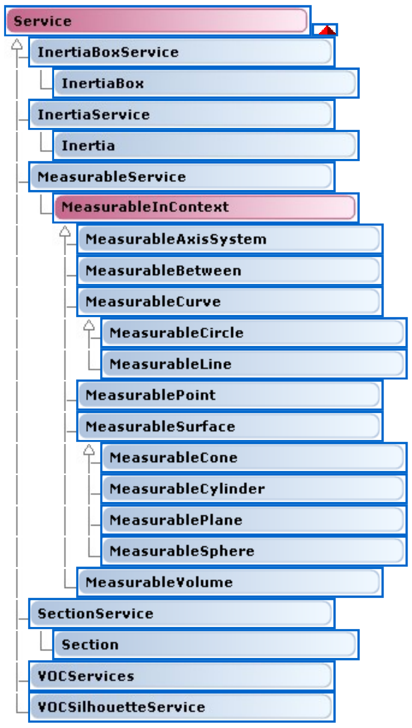
Design Review Object Model Map

See Also Legend



Displaying Inertia Data

This use case shows how to compute the inertia data and the bounding box of a part.

Before you begin:

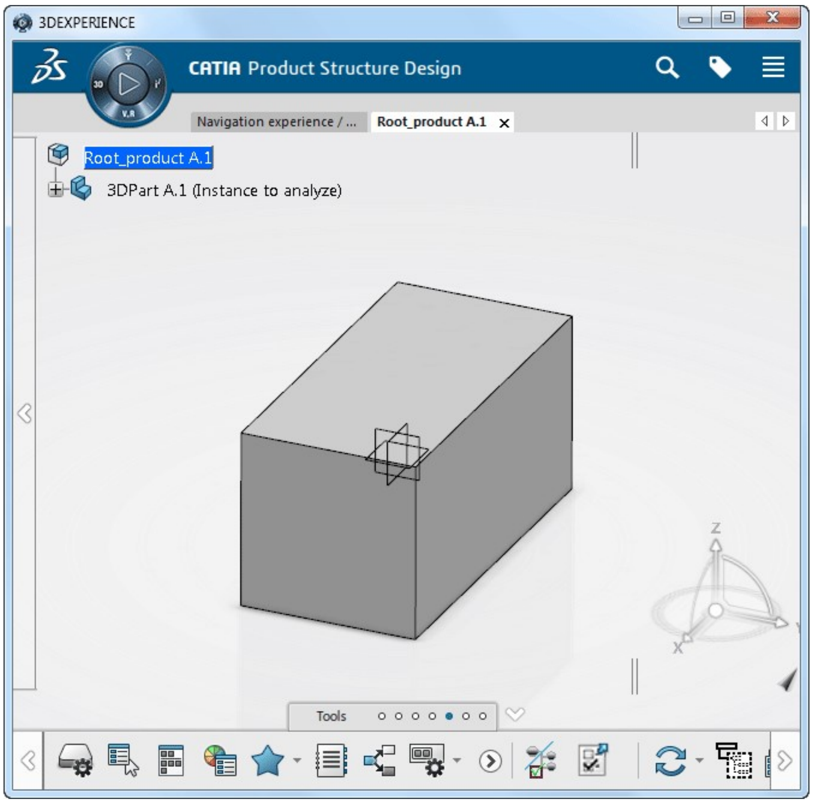
- You should first launch CATIA.
- Import as reference the CAAScdVPMUcInertia.3dxml file from the folder InstallRootFolder\CAADoc\Doc\English\CAAScdVPMDigitalReview where InstallRootFolder is the folder where the API CD-ROM is installed.
- In the Navigation Experience tab, right click Root_product A.1, and click Open.
- In the Root_product A.1 tab, select Root_product A.1 in the tree.
  Note that you can also select the 3D part instance (3DPart A.1), the part body (PartBody), or the pad feature (Pad.1)
- Launch the following macro:

Where to find the macro:

- Display: CAAScdVPMUcInertiaSource.htm
- Run: CAAScdVPMUcInertia.CATScript in InstallRootFolder\CAADoc\Doc\English\CAAScdVPMDigitalReview. Refer to Launching an Automation Use Case.

Fig.1: The Part to Analyze

Related Topics
Design Review Object Model Map


3DEXPERIENCE
CATIA Product Structure Design
Navigation experience / ...
Root_product A.1
Root_product A.1
3DPart A.1 (Instance to analyze)
Tools

This use case can be divided in three steps:

1. Retrieving the Inertia Object
2. Computing the Inertia Data
3. Displaying the Inertia Data

1. **Retrieving the Inertia Objects**

   First, the macro retrieves the active editor and the selection.

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

Dim theSelection As Selection
Set theSelection = theEditor.Selection
...
```

   Then, it retrieves the selected object from the selection.

```
...
If theSelection.Count2 <> 0 Then
  Dim theElement As AnyObject
  Set theElement = theSelection.Item2(1).Value
  ...
```

   It retrieves the inertia services from the editor.

```
   ...
  Dim theInertiaService As InertiaService
  Set theInertiaService = theEditor.GetService("InertiaService")
  Dim theInertiaBoxService As InertiaBoxService
  Set theInertiaBoxService = theEditor.GetService("InertiaBoxService")
  ...
```

   It retrieves the inertia and the inertia box from the inertia services.

```
   ...
  Dim theInertiaElement As Inertia
  Set theInertiaElement = theInertiaService.GetInertiaElement(theElement)
  Dim theInertiaBoxElement As InertiaBox
  Set theInertiaBoxElement = theInertiaBoxService.GetInertiaBoxElement(theElement)
  ...
```

2. **Computing the Inertia Data**

   First, the macro requests to compute the inertia data from the main body only.

```
   ...
  theInertiaElement.OnlyMainBody
  ...
```

   Then it computes the area, the volume, the mass, and the coordnates of the center of gravity.

```
   ...
  Dim theArea As Double
  theArea = theInertiaElement.GetArea

  Dim theVolume As Double
```

```
theVolume = theInertiaElement.GetVolume

Dim theMass As Double
theMass = theInertiaElement.GetMass

Dim theCOG(2) As Double
theInertiaElement.GetCOGPosition theCOG(0), theCOG(1), theCOG(2)
...
```

It computes the inertia matrix, the inertia principle axes, and the inertia moments.

```
 ...
Dim theMatrix(8) As Double
theArea = theInertiaElement.GetInertiaMatrix theMatrix

Dim theAxes(8) As Double
theAxes = theInertiaElement.GetPrincipalAxes theAxes

Dim theMoments(2) As Double
theMoments = theInertiaElement.GetPrincipalMoments theMoments
...
```

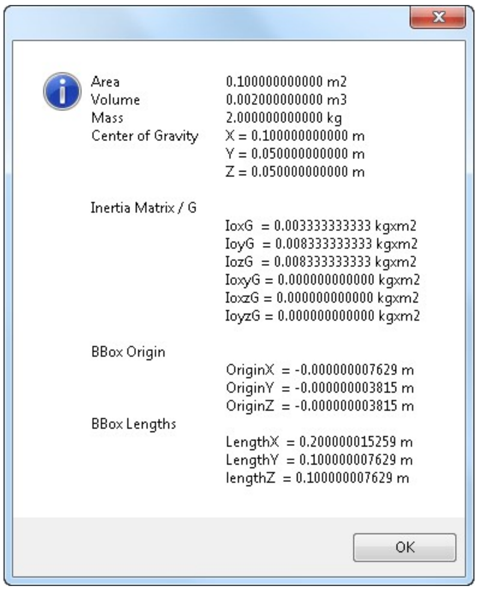
It then computes the bounding box, that is the coordinates of bounding box origin and the lenghes of its sides.

```
 ...
Dim theBoundingBoxOrigin(2) As Double
Dim theBoundingBoxLengths(2) As Double
theInertiaBoxElement.GetBoundingBox theBoundingBoxOrigin, theBoundingBoxLengths
...
```

3. **Displaying the Inertia Data**

   The macro uses the computed inertia data to display it in a message box.

Fig.2: The Inertia Data



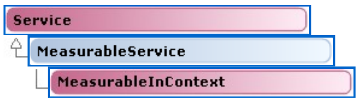
```
Area                0.100000000000 m2
Volume              0.002000000000 m3
Mass                2.000000000000 kg
Center of Gravity   X = 0.100000000000 m
                    Y = 0.050000000000 m
                    Z = 0.050000000000 m

Inertia Matrix / G
                    IoxG  = 0.003333333333 kgxm2
                    IoyG  = 0.008333333333 kgxm2
                    IozG  = 0.008333333333 kgxm2
                    IoxyG = 0.000000000000 kgxm2
                    IoxzG = 0.000000000000 kgxm2
                    IoyzG = 0.000000000000 kgxm2

BBox Origin
                    OriginX  = -0.000000007629 m
                    OriginY  = -0.000000003815 m
                    OriginZ  = -0.000000003815 m
BBox Lengths
                    LengthX  = 0.200000015259 m
                    LengthY  = 0.100000007629 m
                    lengthZ  = 0.100000007629 m
```

In Short

This use case shows how to retrieve the **InertiaService** and **InertiaBoxService** objects from the active **Editor** object. Then it shows you how to retrieve the **Inertia** and **InertiaBox** objects from these service objects. Finally it shows you how to compute the inertia data and the bounding box of the part selected.

MeasurableService Object

See Also Legend Use Cases Properties Methods



Represents a service for measuring purposes.

The **MeasurableService** object lets you retrieve a **MeasurableInContext** object from the selection.

Retrieving the MeasurableService Object

Refer to the Service Object.

Using the MeasurableService Object

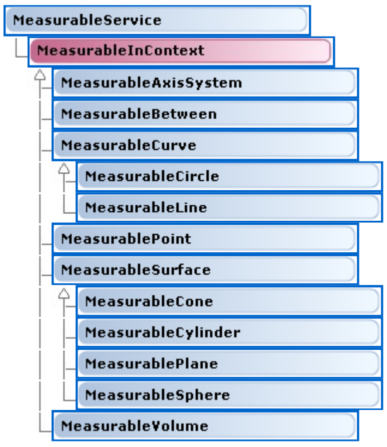
Use the **MeasurableService** object to retrieve the **MeasurableInContext** object you need. For example, to retrieve a **MeasurableCurve** object from the selection:

```
Dim oMeasurableService As MeasurableService
Set oMeasurableService = CATIA.ActiveEditor.GetService("MeasurableService")
Dim oMeasurableCurve As MeasurableCurve
```

```
Set oMeasurableCurve = oMeasurableService.GetMeasurable(oSelection, CAAMeasurableCurve)
```

MeasurableInContext Object

See Also Legend Use Cases Properties Methods



Represents a measurable object.

The **MeasurableInContext** object lets you measure or retrieve data from the selected object. Depending on this selected object type, use the that object deriving from the **MeasurableInContext** object that match either its type or the data you want to measure or retrieve.

Use:

- The **MeasurableAxisSystem** object to retrieve the coordinates of the origin and the components of the vectors of the selected coordinate system expressed in the absolute coordinate system
- The **MeasurableBetween** object to retrieve the angle or the minimum distance between the selected object and another object or a point
- The **MeasurableCurve** object to retrieve the length or the characteristic points of the selected curve
- The **MeasurableCircle** object to retrieve the angle, the axis (normal), the center, or the radius of the selected circle
- The **MeasurableLine** object to retrieve the origin and the direction of the selected line
- The **MeasurablePoint** object to retrieve the coordinates of the selected point
- The **MeasurableSurface** object to retrieve the area, the center of gravity, and the perimeter of the selected surface
- The **MeasurableCone** object to retrieve the angle, the axis, or the two limit points of the selected cone
- The **MeasurableCylinder** object to retrieve the axis, the radius, or the two limit points of the selected cylinder
- The **MeasurablePlane** object to retrieve to retrieve the coordinates of the origin and the components of the vectors expressed in the absolute coordinate system of the selected plane
- The **MeasurableSphere** object to retrieve the center and the radius of the selected sphere
- The **MeasurableVolume** object to retrieve the area, the center of gravity, and the volume of the selected object.

Retrieving the MeasurableInContext Object

Use the **MeasurableService** object to retrieve the **MeasurableInContext** object you need. For example, to retrieve a **MeasurablePoint** object:

```
Dim oMeasurableService As MeasurableService
Set oMeasurableService = CATIA.ActiveEditor.GetService("MeasurableService")
Dim oMeasurablePoint As MeasurablePoint
Set oMeasurablePoint = oMeasurableService.GetMeasurable(oSelection, CAAMeasurablePoint)
```

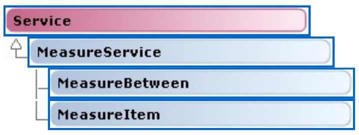
Using the MeasurableInContext Object

Use the **MeasurablePoint** object retrieved above to retrieve the coordinates of the selected point.

```
Dim oMeasurablePoint As MeasurablePoint
Set oMeasurablePoint = ...
Dim XCoordinate As Double
Dim YCoordinate As Double
Dim ZCoordinate As Double
oMeasurablePoint.GetPoint XCoordinate, YCoordinate, ZCoordinate
```

MeasureService Object

See Also Legend Use Cases Properties Methods



Represents a service for measuring purposes.

The **MeasureService** object lets you retrieve a **MeasureBetween** object to measure between two sets of selected objects, or the **MeasureItem** object to measure a set of selected objects.

Retrieving the MeasureService Object

Refer to the Service Object.

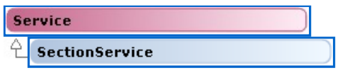
Using the MeasureService Object

Use the **MeasureService** object to retrieve the **MeasureBetween** object:

```
Dim oMeasureService As MeasureService
Set oMeasureService = CATIA.ActiveEditor.GetService("MeasureService")
Dim oMeasureBetween As MeasureBetween
Set oMeasureBetween = oMeasureService.GetMeasureBetween(oSelection1, oSelection2)
...
oMeasureBetween.Compute
```

SectionService Object

See Also Legend Use Cases Properties Methods



Represents a service for sectioning purposes.

The **SectionService** object lets you create and manage **Section** objects.

Retrieving the SectionService Object

Refer to the Service Object.

Using the SectionService Object

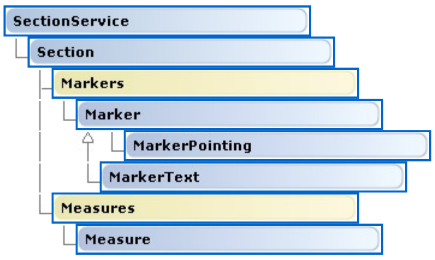
Use the **Add** method of the **SectionService** object to create a **Section** object.

```
Dim oSectionService As SectionService
Set oSectionService = CATIA.ActiveEditor.GetService("SectionService")
Dim oSection As Section
Set oSection = oSectionService.Add
```

To Add a Section, the Product must have a Markup.

Section Object

See Also Legend Use Cases Properties Methods



Represents a section object.

The **Section** object lets you create a section plane sectioning the space. You can retrieve and set the sectioning plane, and export the curves created by the sectioned objects to a 3D part.

Creating a Section Object

Use the **Add** method of the **SectionService** object to create a **Section** object.

```
Dim oSectionService As SectionService
Set oSectionService = CATIA.ActiveEditor.GetService("SectionService")
Dim oSection As Section
Set oSection = oSectionService.Add
```

To Add a Section, the Product must have a Markup.

Positioning the Section Object Plane

Use the **SetPosition** method of the `oSection` **Section** object to set the sectioning plane.

```
Dim aPositioningMatrix(11) As Double
aPositioningMatrix(0)  = 1.0    'X-axis 1st component
aPositioningMatrix(1)  = 0.0    'X-axis 2nd component
aPositioningMatrix(2)  = 0.0    'X-axis 3rd component
aPositioningMatrix(3)  = 0.0    'Y-axis 1st component
aPositioningMatrix(4)  = 1.0    'Y-axis 2nd component
aPositioningMatrix(5)  = 0.0    'Y-axis 3rd component
aPositioningMatrix(6)  = 0.0    'Normal axis 1st component
aPositioningMatrix(7)  = 0.0    'Normal axis 2nd component
aPositioningMatrix(8)  = 1.0    'Normal axis 3rd component
aPositioningMatrix(9)  = 0.0    'Origin X coordinate
aPositioningMatrix(10) = 0.0    'Origin Y coordinate
aPositioningMatrix(11) = 100.0  'Origin Z coordinate

oSection.SetPosition aPositioningMatrix
```

The sectioning plane is defined by two vectors and an origin making up its own coordinate system, and a normal vector.

The sectioning plane is parallel to the XY plane, because its two director vectors are parallel to the X and Y vectors of the absolute coordinate system, and its normal vector is parallel to the Z vector of the absolute coordinate system. Its origin is located at the z position 100.

VOCServices Object

See Also Legend Use Cases Properties Methods

Service
VOCServices

The **VOCServices** object lets you create a Volume from a Reference.

Retrieving the VOCServices Object

To retrieve the VOCServices Object, you need to call the Service from the Representation Reference of a Part.

```
Dim myPart As Part
Set myPart = oEditor.ActiveObject
Dim MyVPMRepRef As VPMRepReference
Set MyVPMRepRef = myPart.Parent

Dim myVOCServices ' As VOCServices
Set myVOCServices = MyVPMRepRef.GetItem("VOCServices")
```

Using the VOCServices Object

Use the **VOCServices** object to create VOCVolumes. For example, to create an Offset Volume of the Root Occurence of an Assembly :

```
Dim oRootOcc As VPMRootOccurrence
Set oRootOcc = ...

Dim oObjVPMReference As VPMReference
Set oObjVPMReference = oRootOcc.ReferenceRootOccurrenceOf

ReDim MyInputProducts(0)
MyInputProducts = Array(oRootOcc)

' Creates a new 3DShape
Dim oNewService As PLMNewService
Set oNewService = CATIA.GetSessionService("PLMNewService")

Dim oEditor As Editor
oNewService.PLMCreate "3DShape", oEditor

' Gets the Part
Dim myPart As Part
Set myPart = oEditor.ActiveObject
Dim MyVPMRepRef As VPMRepReference
Set MyVPMRepRef = myPart.Parent

' Gets the VOC Services
Dim myVOCServices ' As VOCServices
Set myVOCServices = MyVPMRepRef.GetItem("VOCServices")

' Creates the offset Volume
myVOCServices.CreateOffset MyInputProducts, ObjVPMReference, 2#
```

VOCSilhouetteService Object

See Also Legend Use Cases Properties Methods

Service
VOCSilhouetteService

The **VOCSilhouetteService** object lets you create a Silhouette from a Reference.

Retrieving the VOCSilhouetteService Object

To retrieve the VOCSilhouetteService Object, you need to call the Service from the Representation Reference of a Part.

```
Dim myPart As Part
Set myPart = oEditor.ActiveObject
Dim MyVPMRepRef As VPMRepReference
Set MyVPMRepRef = myPart.Parent

Dim myVOCServices ' As VOCSilhouetteService
Set myVOCServices = MyVPMRepRef.GetItem("VOCSilhouetteService")
```

Using the VOCSilhouetteService Object

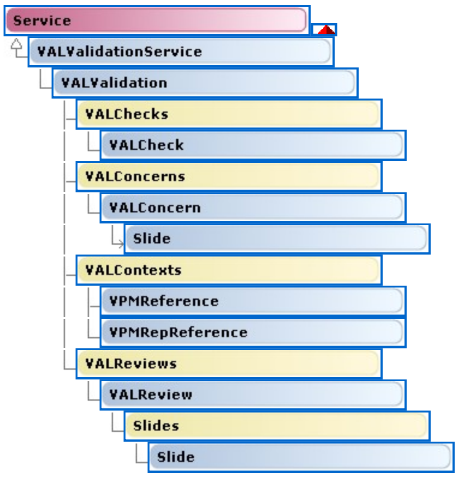
Use the **VOCSilhouetteService** object to create Silhouette Volumes. To create an Silhouette Volume of the Root Occurence of an Assembly :

```
Dim oRootOcc As VPMRootOccurrence
Set oRootOcc = ...

Dim oObjVPMReference As VPMReference
```

```
Set oObjVPMReference = oRootOcc.ReferenceRootOccurrenceOf

ReDim MyInputProducts(0)
MyInputProducts = Array(oRootOcc)

' Gets the current View Point
Dim oViewer As Viewer3D
Set oViewer = CATIA.ActiveWindow.ActiveViewer
Dim viewPoint(0)
Set viewPoint(0) = oViewer.Viewpoint3D

' Creates a new 3DShape
Dim oNewService As PLMNewService
Set oNewService = CATIA.GetSessionService("PLMNewService")

Dim oEditor As Editor
oNewService.PLMCreate "3DShape", oEditor

' Gets the Part
Dim myPart As Part
Set myPart = oEditor.ActiveObject
Dim MyVPMRepRef As VPMRepReference
Set MyVPMRepRef = myPart.Parent

' Gets the VOC Silhouette Service
Dim myVOCServices ' As VOCServices
Set myVOCServices = MyVPMRepRef.GetItem("VOCSilhouetteService")

' Creates the Silhouette Volume
myVOCServices.CreateASilhouette MyInputProducts, ObjVPMReference, viewPoint, 2, 1
```

Design Synthesis Object Model Map

See Also Legend



Creating an Interference Simulation and Processing the Results

This use case primarily focuses on the methodology to create an interference simulation and analyze the results.

Before you begin :

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

Where to find the macro : CAAScdITFUcITFSimCreationSource.htm

This use case:

1. Creating the Simulation
2. Setting the Interferences Groups
3. Executing the Simulation and Computing the Interferences Results
4. Processing the Results

1. **Creating the Simulation**

```
' Gets the Root Reference
Dim RootOcc As VPMRootOccurrence
Set RootOcc = CATIA.ActiveEditor.ActiveObject

Dim RootRef As VPMReference
Set RootRef = RootOcc.ReferenceRootOccurrenceOf

' Gets the Interference Service
Dim ITFServices As InterferenceServices
Set ITFServices = CATIA.ActiveEditor.GetService("InterferenceServices")

' Creates the Simulation and sets its Specification Type and its Interference Groups Computation Type
Dim SpecificationType As CatInterferenceSpecificationType
SpecificationType = catInterferenceSpecificationTypeClash

Dim GroupCmpType2 As CatInterferenceGroupComputationType2
GroupCmpType2 = catInterferenceGroupComputationTypeGroupAgainstGroup

Dim ITFSimu As InterferenceSimulation
```

```
ITFServices.CreateInterferenceSimulation2 RootRef, GroupCmpType2, ITFSimu

 ITFSimu.InterferenceSpecificationType = SpecificationType
...
```

2. **Setting the Interferences Groups**

```
 ...
' Gets the SkateBody Occurence
Dim occSkateBody As VPMOccurrence
Set occSkateBody = RootOcc.Occurrences.Item(1)

' Sets the Skate Body Group
Dim boardGroup As InterferenceGroupObjects
ITFSimu.GetGroupObjects 1, boardGroup

boardGroup.Add2 occSkateBody

' Gets the TruckWheels Occurences
Dim occTruckWheel(1) As VPMOccurrence
Set occTruckWheel(0) = RootOcc.Occurrences.Item(2)
Set occTruckWheel(1) = RootOcc.Occurrences.Item(3)

' Sets the Truck Group
Dim truckGroup As InterferenceGroupObjects
ITFSimu.GetGroupObjects 2, truckGroup

' Creates the Wheel Group
Dim wheelGroup As InterferenceGroupObjects
ITFSimu.AddGroupObjects wheelGroup

' Gets the Trucks Occurrences and the Wheels Occurrences and Adds them to their Group
Dim occTruck As VPMOccurrence
Dim occWheel(1) As VPMOccurrence

For i = 0 To 1
  ' Adds the Truck to the Truck Group
  Set occTruck = occTruckWheel(i).Occurrences.Item(1)
  truckGroup.Add2 occTruck

  ' Adds the Wheels to the Wheel Group
  Set occWheel(0) = occTruckWheel(i).Occurrences.Item(2)
  wheelGroup.Add2 occWheel(0)
  Set occWheel(1) = occTruckWheel(i).Occurrences.Item(3)
  wheelGroup.Add2 occWheel(1)
Next
...
```

The Simulation is created by default with two empty groups. To create the SkateBody Group and the Truck Group, the empty Group are used with the method Ge
create the Wheel Group, the method AddGroupObjects is used.

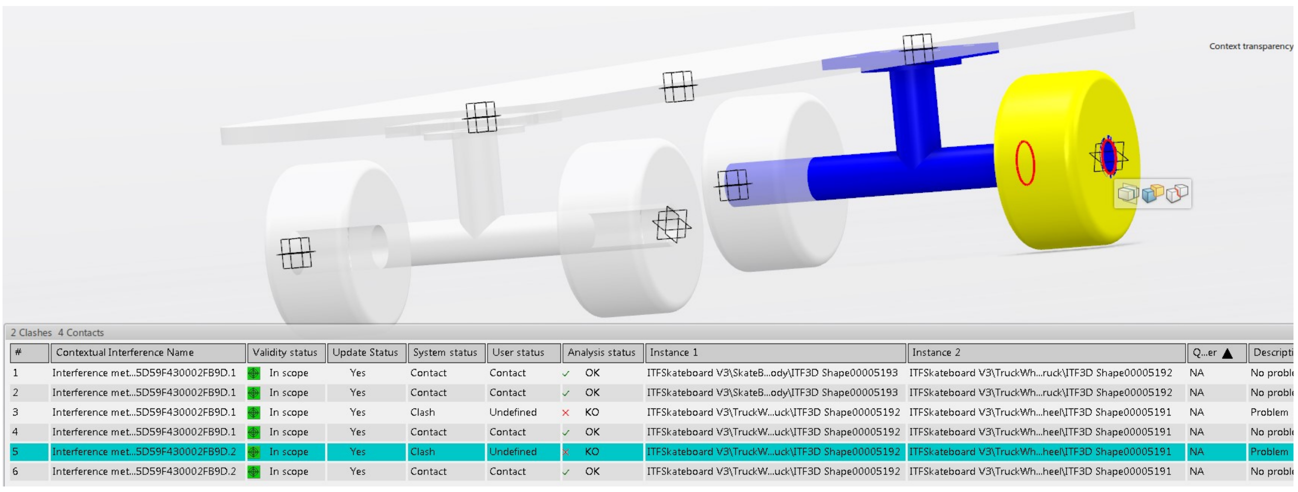
3. **Executing the Simulation and Computing the Interferences Results**

```
...
ITFSimu.Execute
...
```

4. **Processing the Results**

```
' Gets the Interference Results
Dim ListInterferences As InterferenceResults
Set ListInterferences = ITFSimu.InterferenceResults

' Analizes each Result
Dim oInterference As InterferenceResult

For i = 1 To ListInterferences.Count
  Set oInterference = ListInterferences.Item(i)

  If oInterference.Type = catInterferenceResultTypeContact Then
     oInterference.UserType = catInterferenceResultUserTypeContact
     oInterference.AnalysisStatus = catInterferenceResultStatusOK
     oInterference.UserComment = "No problem"
  Else
     oInterference.UserType = catInterferenceResultUserTypeUndefined
     oInterference.AnalysisStatus = catInterferenceResultStatusKO
     oInterference.UserComment = "Problem"
  End If
Next
...
```

Fig.1: The Generated Interference Simulation



2 Clashes 4 Contacts

#	Contextual Interference Name	Validity status	Update Status	System status	User status	Analysis status	Instance 1	Instance 2	Q...er ▲	Descripti
1	Interference met...5D59F430002FB9D.1	🟩 In scope	Yes	Contact	Contact	✓ OK	ITFSkateboard V3\SkateB...ody\ITF3D Shape00005193	ITFSkateboard V3\TruckWh...ruck\ITF3D Shape00005192	NA	No proble
2	Interference met...5D59F430002FB9D.1	🟩 In scope	Yes	Contact	Contact	✓ OK	ITFSkateboard V3\SkateB...ody\ITF3D Shape00005193	ITFSkateboard V3\TruckWh...ruck\ITF3D Shape00005192	NA	No proble
3	Interference met...5D59F430002FB9D.1	🟩 In scope	Yes	Clash	Undefined	✕ KO	ITFSkateboard V3\TruckW...uck\ITF3D Shape00005192	ITFSkateboard V3\TruckWh...heel\ITF3D Shape00005191	NA	Problem
4	Interference met...5D59F430002FB9D.1	🟩 In scope	Yes	Contact	Contact	✓ OK	ITFSkateboard V3\TruckW...uck\ITF3D Shape00005192	ITFSkateboard V3\TruckWh...heel\ITF3D Shape00005191	NA	No proble
5	Interference met...5D59F430002FB9D.2	🟩 In scope	Yes	Clash	Undefined	✕ KO	ITFSkateboard V3\TruckW...uck\ITF3D Shape00005192	ITFSkateboard V3\TruckWh...heel\ITF3D Shape00005191	NA	Problem
6	Interference met...5D59F430002FB9D.2	🟩 In scope	Yes	Contact	Contact	✓ OK	ITFSkateboard V3\TruckW...uck\ITF3D Shape00005192	ITFSkateboard V3\TruckWh...heel\ITF3D Shape00005191	NA	No proble